BYTE

January - March 1987

Including BIX Coverage of the INTEL 80386

LISTINGS SUPPLEMENT

Six great reasons to join BIX today

- **Over 140 *microcomputer-related conferences*:** Join only those subjects that interest you and change selections at any time. Take part when it's convenient for you. Share information, opinions and ideas in focused discussions with other BIX users who share your interests. Easy commands and conference digests help you quickly locate important information.

- ***Monthly conference specials*:** BIX specials connect you with invited experts in leading-edge topics—CD-ROM, MIDI, OS-9 and more. They're all part of your BIX membership.

- ***Microbytes daily*:** Get up-to-the-minute industry news and new product information by joining Microbytes Daily and What's New Hardware and Software.

- ***Public domain software*:** Yours for the downloading, including programs from BYTE articles and a growing library of PD listings.

- ***Electronic mail*:** Exchange private messages with BYTE editors and authors and other BIX users.

- ***Vendor support*:** A growing number of microcomputer manufacturers use BIX to answer your questions about their products and how to use them for peak performance.

What BIX Costs. .How You Pay

ONE-TIME REGISTRATION FEE: $25

Hourly Charges: (Your Time of Access)	**Off-Peak 6PM–7AM** Weekdays Plus Weekends & Holidays	**Peak 7AM–6PM** Weekdays
BIX	$9	$12
Tymnet*	$2	$6
TOTAL	**$11/hr.**	**$18/hr.****

* *Continental U.S. BIX is accessible via Tymnet from throughout the U.S. at charges much less than regular long distance. Call the BIX helpline number listed below for the Tymnet number near you or Tymnet at 1-800-336-0149*

** *User is billed for time on system (i.e., ½ Hr. Off-Peak w/Tymnet = $5.50 charge.)*

BIX *and Tymnet charges billed by* Visa *or* Mastercard *only.*

BIX HELPLINE

(8:30 AM–11:30 PM Eastern Weekdays)

U.S. (except NH)—1-800-227-BYTE
Elsewhere (603) 924-7681

We'll Send You a BIX User's Manual and Subscriber Agreement as Soon as We've Processed Your Registration.
JOIN THE EXCITING WORLD OF BIX TODAY!

JOIN BIX RIGHT NOW:

Set your computer's telecommunications program for full duplex, 8-bit characters, even parity, 1 stop bit OR 7-bit characters, even parity, 1 stop using 300 or 1200 baud.

Call your local Tymnet number and respond as follows:

Tymnet Prompt	You Enter
Garble or "terminal identifier"	a
login:	byteneti <CR>
password:	mgh <CR>
mhis login:	bix <CR>
BIX Logo—Name:	new <CR>

After you register on-line, you're immediately taken to the BIX learn conference and can start using the system right away.

FOREIGN ACCESS:

To access BIX from foreign countries, you must have an account with your local Postal Telephone & Telegraph (PTT) company. From your PTT enter 310600157878. Then enter bix <CR> and new <CR> at the prompts. Call or write us for PTT contact information.

BIX

ONE PHOENIX MILL LANE
PETERBOROUGH, NH 03458
(603) 924-9281

TABLE OF CONTENTS

WELCOME TO BYTE'S QUARTERLY LISTINGS SUPPLEMENT

The BYTE Listings Supplement is produced quarterly as a means of providing interested readers with a printed, source code version of those programs referenced in BYTE articles. It provides a far more extensive look into the techniques of coding and the potentialities of microcomputers than we have space for in each month's BYTE.

Programs contained in this Supplement are referenced by the month the article appeared, the page on which their supporting article begins, and the name of the author who wrote the article.

For those who prefer programs already in electronic format, we have a companion service called Listings on Disk. If you have a modem, listings may be downloaded from the BYTEnet bulletin board and, if you are a member of BIX, the "Listings" area also contains programs referenced in BYTE.

Beginning with this issue of the Supplement, we are also providing a "trailer" section containing material we feel may be of additional interest to BYTE readers. This time, we're including a Best of BIX section chronicling events, facts and opinions surrounding the introduction of Intel's 80386 microprocessor. With succeeding issues we hope to use this section for a variety of informative purposes.

BYTE

SENIOR VICE PRESIDENT/PUBLISHER
Harry L. Brown
EDITOR IN CHIEF
Philip Lemmons

BIX

MANAGING EDITOR, BYTE
Frederic S. Langa

ASSISTANT MANAGING EDITOR
Glenn Hartwig
CONSULTING EDITORS
Steve Ciarcia
Jerry Pournelle
Ezra Shapiro
Bruce Webster
SENIOR TECHNICAL EDITORS
G. Michael Vose, Themes
Gregg Williams, Features
TECHNICAL EDITORS
Dennis Allen
Cathryn Baskin, Reviews
Richard Grehan
Ken Sheldon
George A. Stewart
Jane Morrill Tazelaar
Tom Thompson
Charles D. Weston
Eva White
Stanley Wszola
ASSOCIATE TECHNICAL EDITORS
Curtis Franklin Jr., Best of BIX
Margaret Cook Gurney, Book Reviews

COPY EDITORS
Bud Sadler, Chief
Jeff Edmonds
Nancy Hayes
Cathy Kingery
Margaret A. Richard
Lauren Stickler
Warren Williamson
Judy Winkler

ASSISTANTS
Peggy Dunham, Office Manager
Shawn Allen
L. Ryan McCombs
Jennifer Price
June N. Sheldon

NEWS AND TECHNOLOGY
Gene Smarte, Bureau Chief, Costa Mesa
Jonathan Erickson, Senior Technical Editor, San Francisco
Rich Malloy, Senior Technical Editor, New York
Cindy Kiddoo, Editorial Assistant, San Francisco
ASSOCIATE NEWS EDITORS
Dennis Barker, Microbytes
Anne Fischer Lent, What's New
Stan Miastkowski, What's New
Martha Hicks

CONTRIBUTING EDITORS
Jonathan Amsterdam, programming projects
Mark Dahmke, video, operating systems
Mark Haas, at large
Rik Jadrnicek, CAD, graphics, spreadsheets
Robert T. Kurosaka, mathematical recreations
Phil Lopiccolo, computers in medicine
Alastair J. W. Mayer, software
Alan R. Miller, languages and engineering
Dick Pountain, U.K.
Roger Powell, computers and music
Phillip Robinson, semiconductors

ART
Nancy Rice, Art Director
Joseph A. Gallagher, Assistant Art Director
Jan Muller, Art Assistant
Alan Easton, Drafting

PRODUCTION
David R. Anderson, Production Director
Denise Chartrand
Michael J. Lonsky
Virginia Reardon

TYPOGRAPHY
Sherry McCarthy, Chief Typographer
Donna Sweeney
Selinda Chiquoine

EXECUTIVE EDITOR, BIX
George Bond

SENIOR EDITOR
David Betz
ASSOCIATE EDITORS
Tony Lockwood
Donna Osgood, San Francisco
MICROBYTES DAILY
Dennis Barker, Coordinator, Peterborough
Gene Smarte, Bureau Chief, Costa Mesa
Stan Miastkowski, Peterborough
Rick Cook, Phoenix
Jonathan Erickson, San Francisco
Martha Hicks, Peterborough
Cindy Kiddoo, San Francisco
Anne Fischer Lent, Peterborough
Rich Malloy, New York
Stan Miastkowski, Peterborough
Lynne Nadeau, Peterborough
Jeff Merron, Peterborough
Wayne Rash, Washington, DC
William Webb, Peterborough
Lamont Wood, San Antonio
GROUP MODERATORS
David Allen, Applications
Frank Boosman, Artificial Intelligence
Leroy Casterline, Other
Marc Greenfield, Programming Languages
Jim Howard, Graphics
Gary Kendall, Operating Systems
Steve Krenek, Computers
Brock Meeks, Telecommunications
Barry Nance, New Technology
Donald Osgood, Computers
Sue Rosenberg, Other
Jon Swanson, Chips
BUSINESS AND MARKETING
Doug Webster, Director (603-924-9027)
Patricia Bausum, Secretary
Denise A. Greene, Customer Service
Brian Warnock, Customer Service
Tammy Burgess, Customer Credit and Billing
TECHNOLOGY
Clayton Lisle, Director, Business Systems Technology, MHIS
Bill Garrison, Business Systems Analyst
Jack Reilly, Business Systems Analyst
Linda Wolff, Senior Business Systems Analyst

Officers of McGraw-Hill Information Systems Company: President: Richard B. Miller. Executive Vice Presidents: Frederick P. Jannott, Construction Information Group; Russell C. White, Computers and Communications Information Group; J. Thomas Ryan, Marketing and International. Senior Vice Presidents: Francis A. Shinal, Controller; Robert C. Violette, Manufacturing and Technology. Senior Vice Presidents and Publishers: Laurence Altman, Electronics; Harry L. Brown, BYTE; David J. McGrath, Construction Publications. Group Vice President: Peter B. McCuen, Communications Information. Vice President: Fred O. Jensen, Planning and Development.

Officers of McGraw-Hill, Inc.: Harold W. McGraw, Jr., Chairman; Joseph L. Dionne, President and Chief Executive Officer; Robert N. Landes, Executive Vice President and Secretary; Walter D. Serwatka, Executive Vice President and Chief Financial Officer; Shel F. Asen, Senior Vice President, Manufacturing; Robert J. Bahash, Senior Vice President, Finance and Manufacturing; Ralph R. Schulz, Senior Vice President, Editorial; George R. Elsinger, Vice President, Circulation; Ralph J. Webb, Vice President and Treasurer.

BYTE, BYTE, and The Small Systems Journal are registered trademarks of McGraw-Hill Inc.

EDITORIAL AND BUSINESS OFFICE: One Phoenix Mill Lane, Peterborough, New Hampshire 03458, (603) 924-9281.

West Coast Offices: 425 Battery St., San Francisco, CA 94111, (415) 954-9718; 3001 Red Hill Ave., Building -1, Suite 222, Costa Mesa, CA 92626, (714) 557-6292. **New York Editorial Office:** 1221 Avenue of the Americas, New York, NY 10020, (212) 512-3175.

BYTEnet: (617) 861-9764 (set modem at 8-1-N or 7-1-E; 300 or 1200 baud).

BYTE (ISSN 0360-5280) is published monthly with one extra issue per year by McGraw-Hill Inc. Founder: James H. McGraw (1860-1948). Executive, editorial, circulation, and advertising offices: One Phoenix Mill Lane, Peterborough, NH 03458, phone (603) 924-9281. Office hours: Monday through Thursday 8:30 AM — 4:30 PM, Friday 8:30 AM — 1:00 PM, Eastern Time. Address subscriptions to BYTE Subscriptions, P.O. Box 590, Martinsville, NJ 08836. Postmaster: send address changes, USPS Form 3579, undeliverable copies, and fulfillment questions to BYTE Subscriptions, P.O. Box 596, Martinsville, NJ 08836. Second-class postage paid at Peterborough, NH 03458 and additional mailing offices. Postage paid at Winnipeg, Manitoba. Registration number 9321. Subscriptions are $22 for one year, $40 for two years, and $58 for three years in the U.S. and its possessions. In Canada and Mexico, $25 for one year, $45 for two years, $65 for three years. $69 for one year air delivery to Europe. 31,000 yen for one year air delivery to Japan, 15,600 yen for one year surface delivery to Japan, $37 surface delivery elsewhere. Air delivery to selected areas at additional rates upon request. Single copy price is $3.50 in the U.S. and its possessions, $4.25 in Canada and Mexico, $4.50 in Europe, and $5 elsewhere. Foreign subscriptions and sales should be remitted in U.S. funds drawn on a U.S. bank. Please allow six to eight weeks for delivery of first issue. Printed in the United States of America.

Address editorial correspondence to: Editor, BYTE, One Phoenix Mill Lane, Peterborough, NH 03458. Unacceptable manuscripts will be returned if accompanied by sufficient postage. Not responsible for lost manuscripts or photos. Opinions expressed by the authors are not necessarily those of BYTE.

BYTE is available in microform from University Microfilms International, 300 North Zeeb Rd., Dept. PR, Ann Arbor, MI 48106 or 18 Bedford Row, Dept. PR, London WC1R 4EJ, England.
Subscription questions or problems should be addressed to: BYTE Subscriber Service, P.O. Box 328, Hancock, NH 03449.

ABC

INDEX

January

February

March

Best of BIX

January

PAL1.BAS
Contributed by: Robert A. Freedman
"Getting Started with PALs," by Robert A. Freedman. January, page 223.

```
10 REM DEFINE VARIABLEs
20 CLEAR
30 DEFINT C,F,H,N,O,T,W,X,Y,Z,L
40 DIM F(39,79),N2(24)
50 DEFSTR A,D,P
60 DIM A(15),P(25),N(12),N1(12)
70 X$=CHR$(32):A=STRING$(50," "):AT=A+" "
80 C=0:X=0:Y=0:Z=0:S$=""
90 REM ***********************************************************
100 REM ***    INFORMATION ABOUT PALASM SPECEFICATION           ***
110 REM ***********************************************************
115 PRINT"(c) Copyright 1983 Monolithic Memories Inc. All Rights Reserved"
116 PRINT
120 PRINT TAB(13)"PALASM-20/24 in Basic":PRINT
130 PRINT TAB(11)"Revision level 1.2"
140 PRINT TAB(11)"07/15/81 D. Jones"
150 PRINT TAB(11)"06/22/83 U. Mueller & C.B. Lee"
160 PRINT
170 PRINT"Note: When using the 20X- Pals in the series 24"
180 PRINT"family, the XOR operator ':+:' should start a new"
190 PRINT"line.  Thus:  /Q1 := A*B + C*D :+: E*F + G*H"
200 PRINT"is an error":PRINT
210 PRINT"It should read:"
220 PRINT"      /Q1 :=  A*B + C*D     or     /Q1 :=  A*B"
230 PRINT"          :+: E*F + G*H                 +  C*D"
240 PRINT"                                       :+: E*F"
250 PRINT"The second format is recommended        +  G*H"
260 PRINT"for ease of reading and commenting."
270 PRINT"Note also a space is required before and after"
280 PRINT"the '+' in the first format."
285 PRINT
290 PRINT"Press a key to continue..."
300 DUMMY$=INKEY$:IF DUMMY$="" THEN 300
310 FOR I=1 TO 23:PRINT:NEXT
320 PRINT"What is your input file name ?";
330 LINE INPUT F$:IF F$="" THEN 120:REM * GET FILENAME *
340 X=1:OPEN "I",1,F$:REM * X=NUMBER OF LINES READ IN *
350 FOR I=1 TO 10:PRINT:NEXT
360 PRINT"           ASSEMBLING...PLEASE WAIT !!!"
370 PRINT:PRINT
380 REM ***********************************************************
390 REM ***    VERIFY PART NUMBER AND GET TYPE                  ***
400 REM ***********************************************************
410 LINE INPUT #1,A:TY=0
415 IF A="" THEN 410
420 X=INSTR(A,"PAL")
430 OT$=MID$(A,X+5,1):P=MID$(A,X+6,2):NO=VAL(P)
440 PN=MID$(A,X,8):IF RIGHT$(PN,1)=" " THEN PN=LEFT$(PN,7)
450 P=LEFT$(PN,3):IF P<>"PAL" THEN GOTO 590 ELSE P=MID$(PN,4,5)
460 OPEN "I",2,"PALTABLE.DAT"
465 INPUT #2,TYPE$
470 IF TYPE$<>P THEN LINE INPUT #2,DUMMY$:GOTO 465
475 INPUT #2,TY,XM,YM,S,FCODE
485 FOR I=1 TO S
495 INPUT #2,N2(I)
505 NEXT I
515 FOR I=0 TO S-12
525 INPUT #2,N(I),N1(I)
535 NEXT I
540 FOR I=1 TO INT((S/2)-1)
545 INPUT #2,IX(I)
```

continued

```
550 NEXT I
555 CLOSE 2
590 IF TY=0 THEN GOSUB 2020:PRINT"INVALID PART NUMBER":END
600 PRINT"PART NUMBER ... OK !!!"
605 GOSUB 3000
610 REM ***********************************************************
620 REM ***     VERIFY PIN LIST                                ***
630 REM ***********************************************************
640 FOR I=1 TO 4:LINE INPUT #1,A:NEXT I
650 Y=1
660 A=A+" ":C=LEN(A):FOR X=1 TO C
670 P=MID$(A,X,1):IF P<>" " THEN P(Y)=P(Y)+P
680 IF P=" " AND P(Y)<>"" THEN Y=Y+1
690 NEXT:IF Y=S+2 THEN 710 ELSE IF Y<S+2 THEN LINE INPUT #1,A:GOTO 660
700 GOSUB 2020:PRINT"INVALID PIN LIST":END
710 W=(S+1)/2:IF P(W)="GND" THEN 730
720 PRINT"ERROR CORRECTED... PIN";W;" IS NOW 'GND'":P(W)="GND"
730 W=S+1:IF P(W)="VCC" THEN 750
740 PRINT"ERROR CORRECTED... PIN";W;" IS NOW 'VCC'":P(W)="VCC"
750 PRINT"PIN LIST ...... OK !!!"
780 REM ***********************************************************
790 REM ***      FIND OUTPUT IN EQUATION                       ***
800 REM ***********************************************************
810 OU=0:IF TY>4 AND TY<9 THEN NO=8
820 IF TY=16 THEN NO=8 ELSE IF TY=15 THEN NO=10
830 LINE INPUT #1,A:IF EOF(1) THEN CLOSE:GOTO 2380
840 IF LEFT$(A,1)=";" OR INSTR(A,"=")=0 THEN 830
850 ZZ=INSTR(A,";"):IF ZZ<>0 THEN A=LEFT$(A,ZZ-1)
860 IF INSTR(A," ")=0 THEN 880
870 ZZ=INSTR(A," "):A=LEFT$(A,ZZ-1)+RIGHT$(A,LEN(A)-ZZ):GOTO 860
880 AA=A:FC=0:FS=0:FR=0:AT="":DL=")/ "
890 CE=INSTR(A,"="):IF CE=0 THEN 830
900 OU=OU+1:IF OU>NO THEN 1650
910 AL=LEFT$(A,CE-1):CT=LEN(A):CN=CE
920 CN=CN-1:IF CN=0 THEN GOTO 950
930 P=MID$(A,CN,1):IF P=" " THEN 920 ELSE IF P=":" THEN FR=1:GOTO 920
940 P=MID$(A,CN,1):IF INSTR(DL,P)=0 THEN AT=P+AT:CN=CN-1:IF CN<>0 THEN 940
950 IF INSTR(AT," ")<>0 THEN AT=LEFT$(AT,LEN(AT)-1):GOTO 950
960 FOR Z=12 TO S:IF AT=P(Z) OR P(Z)=("/"+AT) THEN GOSUB 1910:GOTO 990
970 IF AT=("/"+P(Z)) THEN GOSUB 1910:GOTO 990 ELSE NEXT
980 GOSUB 2020:PRINT"OUTPUT UNDEFINED BY PIN LIST":GOTO 1680
990 IF Y=0 THEN GOSUB 2020:PRINT"INVALID OUTPUT PIN":GOTO 1680
1000 IF Y>100 THEN FR=1:Y=Y-100 ELSE IF Y<0 THEN FC=1:Y=-Y ELSE FS=1
1010 Y=Y-1:PRINT"ASSEMBLING OUTPUT:  ";P(Z);" ;PL =";Y;"      "
1030 Y1=Y+NP:GOSUB 1720
1040 IF (FS=1 OR FR=1) AND INSTR(AL,")")<>0 THEN 1070
1050 IF FC=1 AND INSTR(AL,")")=0 THEN Y=Y+1:CN=CE+1:GOSUB 1720:GOTO 1350
1060 IF FC=1 THEN 1120 ELSE CN=CE+1:GOTO 1350
1070 GOSUB 2020:PRINT"EQUATION INVALID FOR THIS OUTPUT TYPE"
1080 PRINT"-->";A;" PIN =";ZO:END
1090 REM ***********************************************************
1100 REM ***    THREE-STATE ENABLE ONLY                         ***
1110 REM ***********************************************************
1120 IF INSTR(AL,"VCC")<>0 THEN CN=CE+1:Y=Y+1:GOSUB 1720:GOTO 1350
1130 CN=INSTR(AL,"("):CT=INSTR(AL,")"):IF CN=0 OR CT=0 THEN 1070
1140 A=AL:CN=CN+1:CT=CT-1
1150 IF INSTR(A,"+")=0 THEN 1170
1160 GOSUB 2020:PRINT"INVALID CONDITIONAL STATEMENT":PRINT"-->";A:END
1170 DL="(:)+*":AT=""
1180 IF CN>CT THEN GOTO 1220
1190 P=MID$(A,CN,1):IF P=" " THEN CN=CN+1:GOTO 1180
1200 IF INSTR(DL,P)=0 THEN AT=AT+P:IF CN<>CT THEN CN=CN+1:GOTO 1180
1210 GOSUB 1560:GOTO 1170
1220 Y=Y+1:A=AA:CN=CE+1:CT=LEN(A)
1230 GOSUB 1720
1240 GOTO 1350
1250 REM ***********************************************************
1260 REM ***    INPUT PROCESSING FOR SIMPLE OUTPUTS             ***
1270 REM ***********************************************************
1280 LINE INPUT #1,A:IF EOF(1) THEN CLOSE:GOTO 2380
1290 IF INSTR(A,"DESCRIPTION")<>0 THEN 2380
1300 IF INSTR(A,"FUNCTION TABLE")<>0 THEN 2380
1310 ZZ=INSTR(A,";"):IF ZZ<>0 THEN A=LEFT$(A,ZZ-1)
1320 IF INSTR(A," ")=0 THEN 1340
1330 ZZ=INSTR(A," "):A=LEFT$(A,ZZ-1)+RIGHT$(A,LEN(A)-ZZ):GOTO 1320
1340 CT=LEN(A):CN=1:IF INSTR(A,"=")<>0 THEN 880
1350 AT="":P=MID$(A,CN,1):IF P<>"+" THEN 1370
```

```
1360 GOSUB 1560:Y=Y+1:GOSUB 1720:GOTO 1350
1370 IF P<>":" THEN 1390 ELSE IF MID$(A,CN,3)<>":+:" THEN 1390
1380 GOSUB 1560:CN=CN+2:Y=2*INT((Y+2)/2):GOSUB 1720:GOTO 1350
1390 IF P="*" THEN GOSUB 1560:GOTO 1350
1400 IF TY=7 AND (P="(" OR P=")") THEN 2040
1410 IF P="(" OR P=")" OR (P=":" AND TY<>15) THEN 1070
1420 CO=INSTR(CN,A,"+")
1430 CA=INSTR(CN,A,"*")
1440 IF CO>0 AND CA>0 AND CA>CO THEN CD=CO:GOTO 1480
1450 IF CO>0 AND CA=0 THEN CD=CO:GOTO 1480
1460 CD=CA
1470 IF CD=0 THEN CD=CT+1
1480 AT=MID$(A,CN,CD-CN):GOSUB 1560:CN=CD:IF CN=CO THEN Y=Y+1:GOSUB 1720
1490 CN=CD+1:IF CD>CT THEN 1280
1500 GOTO 1350
1510 GOSUB 2020:PRINT"EXCESSIVE NUMBER OF TERMS FOR THIS OUTPUT"
1520 PRINT"MAXIMUM NUMBER OF TERMS IS";NP;"FOR OUTPUT PIN";ZO:END
1530 REM *******************************************************
1540 REM ***   INPUT MATCH AND SET FUSE                      ***
1550 REM *******************************************************
1560 IF AT="" THEN CN=CN+1:RETURN
1570 FOR Z=1 TO S+1
1580 IF AT=P(Z) THEN GOSUB 1670:X=X-1:GOTO 1640
1590 IF AT="/"+P(Z) THEN GOSUB 1670:GOTO 1640
1600 IF ASC(P(Z))=47 AND AT=MID$(P(Z),2) THEN GOSUB 1670:GOTO 1640
1610 NEXT
1620 IF LEFT$(AT,5)="CARRY" THEN 1280
1630 GOSUB 2020:PRINT"INPUT UNDEFINED BY PIN LIST":GOTO 1680
1640 F(X,Y)=0:NB=NB-1:CN=CN+1:RETURN
1650 GOSUB 2020:PRINT"EXCESSIVE NUMBER OF EQUATIONS GIVEN."
1660 PRINT"ONLY THE FIRST";NO;" WILL BE ASSEMBLED.":GOTO 2380
1670 X=N2(Z):IF X<>0 THEN RETURN ELSE GOSUB 2020:PRINT"INVALID INPUT PIN"
1680 PRINT"-->";A;"  >";AT;"<":END
1690 REM *******************************************************
1700 REM ***      INITZL PROD LINE WITH BLOWN FUSES          ***
1710 REM *******************************************************
1720 IF Y>Y1 THEN 1510
1730 FOR I=0 TO XM:IF F(I,Y)=0 THEN F(I,Y)=1:NB=NB+1
1740 NEXT:RETURN
1910 Y=N(Z-12):NP=N1(Z-12):RETURN
2020 PRINT"*** ERROR ***":RETURN
2030 REM *******************************************************
2040 REM ***       FOR 16A4 AND 16X4 PALS ONLY               ***
2050 REM *******************************************************
2060 IF P=":"THEN A1=MID$(A,CN,3)ELSE GOTO 2100
2070 IF A1=":+:"THEN Y=4*(INT((Y+4)/4)):GOSUB 1720:CN=CN+3:GOTO 1390
2080 IF A1=":*:"THEN GOSUB 2020:PRINT"':*:' IS USED INSIDE PARENTHESES
ONLY":END
2090 GOSUB 2020:PRINT">";P;"< IS INVALID AS USED IN:":PRINT"-->";A:END
2100 N8=CN:N9=INSTR(CN,A,")"):IF N9=0 THEN 2090
2110 A1=MID$(A,N8+1,N9-N8-1)
2120 N=VAL(RIGHT$(A1,1)):IF N<0 OR N>3 THEN 2130 ELSE 2140
2130 GOSUB 2020:PRINT"INVALID EXPRESSION '";A1;"'":END
2140 X=N*4+8
2150 N0=LEN(A1)-1:IF N0>6 THEN 2130
2160 ON N0 GOTO 2170,2190,2210,2220,2240,2290
2170 C=2:GOSUB 2340:IF MID$(A1,1,1)="A"THEN C=3 ELSE C=0
2180 GOSUB 2340:GOTO 2330
2190 C=1:GOSUB 2340:IF MID$(A1,2,1)="A"THEN C=0 ELSE C=3
2200 GOSUB 2340:GOTO 2330
2210 AT=A1:GOTO 1630
2220 C=2:GOSUB 2340:IF INSTR(A1,"+")<>0 THEN 2330
2230 C=0:GOSUB 2340:C=3:GOSUB 2340:GOTO 2330
2240 IF INSTR(A1,"+B")<>0 THEN C=0:GOSUB 2340:GOTO 2330
2250 IF INSTR(A1,"+/")<>0 THEN C=3:GOSUB 2340:GOTO 2330
2260 C=1:GOSUB 2340:C=2:GOSUB 2340
2270 IF INSTR(A1,"*B")<>0 THEN C=0:GOSUB 2340:GOTO 2330
2280 C=3:GOSUB 2340:GOTO 2330
2290 IF INSTR(A1,"+/")<>0 THEN C=1:GOSUB 2340:GOTO 2330
2300 IF INSTR(A1,"+:")<>0 THEN C=1:GOSUB 2340:C=2:GOSUB 2340:GOTO 2330
2310 C=0:GOSUB 2340:C=3:GOSUB 2340
2320 IF INSTR(A1,"*/")<>0 THEN C=1:GOSUB 2340:GOTO 2330
2330 CN=N9+1:GOTO 1350
2340 F(X+C,Y)=0:NB=NB-1:RETURN
```

continued

```
2350 REM ****************************************************
2360 REM ***     SAVE VARIABLES  &  CHAIN NEXT PRG.       ***
2370 REM ****************************************************
2380 CLOSE
2382 FOR I=12 TO S
2384 IF N(I-12)<0 THEN N(I-12)=-N(I-12)
2386 IF N(I-12)>100 THEN N(I-12)=N(I-12)-100
2387 IF N(I-12)=0 THEN 2389
2388 N(I-12)=(N(I-12)-1)+N1(I-12)-1
2389 NEXT I
2390 OPEN "O",1,"PALTEMP.DAT"
2400 WRITE #1,TY,FCODE,TYPE$
2410 WRITE #1,NB,S,XM,YM,F$
2420 WRITE #1,OT$
2430 FOR J=0 TO YM
2440 A=""
2450 FOR I=0 TO XM STEP 2
2460 A=A+RIGHT$(STR$(F(I,J)),1)+RIGHT$(STR$(F(I+1,J)),1)
2470 NEXT I
2480 PRINT #1,A
2490 PRINT J;" ";CHR$(13);
2500 NEXT J
2510 FOR I=0 TO S-12
2520 WRITE #1,N(I),N1(I)
2530 NEXT I
2532 FOR I=1 TO S+1
2534 WRITE #1,P(I)
2536 NEXT I
2540 CLOSE
2550 RUN "PAL2"
2560 END
3000 C=0:FOR L1=1 TO INT((S/2)-1)
3010 RESTORE
3020 FOR L2=1 TO IX(L1)-1
3030 READ IN,IN,IN,IN,IN,IN,IN,IN
3040 NEXT L2
3050 FOR L2=1 TO 8
3060 READ IN
3070 ON IN GOSUB 3150,3200,3250,3300,3350,3400,3450
3075 C=C+1
3080 NEXT L2
3090 NEXT L1
3100 RETURN
3150 RETURN
3200 FOR I=0 TO XM
3210 F(I,C)=3
3220 NEXT I
3230 RETURN
3250 FOR I=0 TO XM
3260 F(I,C)=2
3270 NEXT I
3280 RETURN
3300 FOR I=6 TO XM-5 STEP 4
3310 F(I,C)=3:F(I+1,C)=3
3320 NEXT I
3330 RETURN
3350 FOR I=10 TO XM-9 STEP 4
3360 F(I,C)=3:F(I+1,C)=3
3370 NEXT I
3380 RETURN
3400 FOR I=14 TO XM-13 STEP 4
3410 F(I,C)=3:F(I+1,C)=3
3420 NEXT I
3430 RETURN
3450 FOR I=18 TO XM-17 STEP 4
3460 F(I,C)=3:F(I+1,C)=3
3470 NEXT I
3480 RETURN
5000 DATA 1,1,1,1,1,1,1,1
5010 DATA 2,2,2,2,2,2,2,2
5020 DATA 3,3,3,3,3,3,3,3
5030 DATA 4,4,3,3,3,3,3,3
5040 DATA 5,5,3,3,3,3,3,3
5050 DATA 5,5,5,5,3,3,3,3
5060 DATA 6,6,6,6,3,3,3,3
5070 DATA 6,6,3,3,3,3,3,3
5080 DATA 7,7,7,7,7,7,3,3
```

```
5090 DATA 7,7,7,7,3,3,3,3
5100 DATA 1,1,1,1,3,3,3,3
```

PAL2.BAS
Contributed by: Robert A. Freedman
"Getting Started with PALs," by Robert A. Freedman. January, page 223.

```
10 REM DEFINE VARIABLEs
20 DEFINT C,F,H,N,O,T,W,X,Y,Z,L
30 DEFSTR A,D,P
40 DIM F(39,79),A(31),P(24),N(12),N1(12)
50 C=0:X=0:Y=0:Z=0:S$=""
60 OPEN "I",1,"PALTEMP.DAT"
70 INPUT #1,TY,FCODE,TYPE$
80 INPUT #1,NB,S,XM,YM,F$,OT$
90 FOR J=0 TO YM
100 INPUT #1,A
110 PRINT J;" ";CHR$(13);
120 FOR I=0 TO XM
130 F(I,J)=VAL(MID$(A,I+1,1))
140 NEXT I
150 NEXT J
160 FOR I=0 TO S-12
170 INPUT #1,N(I),N1(I)
180 NEXT I
190 FOR I=1 TO S+1
200 INPUT #1,P(I)
210 NEXT I
220 CLOSE
230 GOTO 760
240 REM **********************************************************
250 REM ***       HEX FUNCTION                                 ***
260 REM **********************************************************
270 GOSUB 3120
280 IF TY>7 THEN 330
290 A(0)="0":A(1)="1":A(2)="2":A(3)="3":A(4)="4":A(5)="5":A(6)="6"
300 A(7)="7":A(8)="8":A(9)="9":A(10)="A":A(11)="B":A(12)="C":A(13)="D"
310 A(14)="E":A(15)="F"
320 GOTO 390
330 A(0)="00":A(1)="01":A(2)="02":A(3)="03":A(4)="04":A(5)="05"
340 A(6)="06":A(7)="07":A(8)="08":A(9)="09":A(10)="0A":A(11)="0B"
350 A(12)="0C":A(13)="0D":A(14)="0E":A(15)="0F":A(16)="10":A(17)="11"
360 A(18)="12":A(19)="13":A(20)="14":A(21)="15":A(22)="16":A(23)="17"
370 A(24)="18":A(25)="19":A(26)="1A":A(27)="1B":A(28)="1C":A(29)="1D"
380 A(30)="1E":A(31)="1F"
390 FOR Y=0 TO (S+S+1)
400 A=""
410 IF Y=8 THEN Y=(S+S)-6
420 FOR X=0 TO XM
430 IF S=23 AND X=20 THEN GOSUB 520:A=""
440 H=(F(X,Y) AND 1)+(2*(F(X,Y+8) AND 1))+(4*(F(X,Y+16) AND 1))+(8*(F(X,Y+24)
AND 1))
450 IF S=19 THEN 470
460 H=H+(16*(F(X,Y+32) AND 1))
470 A=A+A(H)+" "
480 NEXT X
490 GOSUB 520
500 NEXT Y
510 GOTO 1720
520 IF X$="" THEN 540
530 PRINT #1,A
540 PRINT A
550 RETURN
560 REM **********************************************************
570 REM ***      BHLF AND BPNF FUNCTIONS                       ***
580 REM **********************************************************
590 GOSUB 3120
600 FOR Y=0 TO (S+S+1)
610 A=""
620 IF Y=8 THEN Y=(S+S)-6
630 FOR X=0 TO XM
634 IF S=19 AND (X=8 OR X=16 OR X=24) THEN GOSUB 520:A=""
```

continued

```
636 IF S=23 AND (X=10 OR X=20 OR X=30) THEN GOSUB 520:A=""
640 IF (F(X,Y) AND 1)=1 THEN H$=D1 ELSE H$=D0
650 IF (F(X,Y+8) AND 1)=1 THEN H$=D1+H$ ELSE H$=D0+H$
660 IF (F(X,Y+16) AND 1)=1 THEN H$=D1+H$ ELSE H$=D0+H$
670 IF (F(X,Y+24) AND 1)=1 THEN H$=D1+H$ ELSE H$=D0+H$
680 IF S=19 THEN 700
690 IF (F(X,Y+32) AND 1)=1 THEN H$=D1+H$ ELSE H$=D0+H$
700 A=A+"b"+H$+"f "
710 NEXT X
720 GOSUB 520
730 NEXT Y
740 GOTO 1720
750 GOSUB 3010:PRINT"INVALID FILENAME":END
760 REM **********************************************************
770 REM ***     OPTION SELECT                                  ***
780 REM **********************************************************
790 CLOSE:PRINT"(C) Copyright 1983 Monolithic Memories Inc. All rights
Reserved"
795 PRINT:PRINT:PRINT:PRINT:PRINT
800 PRINT"Enter option at this time:":PRINT
810 PRINT"X - Xplot"
820 PRINT"B - Brief Xplot"
830 PRINT"H - Hex (Ascii Hex programmer format)"
840 PRINT"N - BPNF (Ascii programmer format)"
850 PRINT"L - BHLF (Ascii programmer format)"
860 PRINT"P - Program pal (SD20/24 Format)"
870 PRINT"O - Pinout"
880 PRINT"J - Jedec format"
890 PRINT"E - Echo (Reprints pal design spec.)"
900 PRINT"R - Restart Pal-assembler"
910 PRINT"Q - Quit (End program)"
920 PRINT:PRINT:PRINT:PRINT:PRINT"OPTION (without Return) ?";
930 S$=INKEY$:IF S$="" THEN 930
935 PRINT S$
940 IF S$="X" OR S$="x"THEN GOSUB 1360:GOTO 790
950 IF S$="B" OR S$="b"THEN GOSUB 1360:GOTO 790
960 IF S$="H" OR S$="h"THEN GOSUB 270:GOTO 790
970 IF S$="N" OR S$="n"THEN D0="N":D1="P":GOSUB 590:GOTO 790
980 IF S$="L" OR S$="l"THEN D0="L":D1="H":GOSUB 590:GOTO 790
990 IF S$="P" OR S$="p"THEN GOSUB 1090:GOTO 790
1000 IF S$="E" OR S$="e" THEN GOSUB 3250:GOTO 790
1010 IF S$="J" OR S$="j" THEN GOSUB 1820:GOTO 790
1030 IF S$="O" OR S$="o" THEN GOSUB 3340:GOTO 790
1040 IF S$="R" OR S$="r"THEN RUN "PAL1"
1050 IF S$="Q" OR S$="q"THEN PRINT:END ELSE 930
1060 REM **********************************************************
1070 REM ***     DATA TO PROGRAMMER IN SD20/24 FORMAT           ***
1080 REM **********************************************************
1090 PRINT:PRINT
1100 PRINT:PRINT:INPUT" Is programmer 'ON' and 'RESET' (Y/N) ";DU$
1110 IF LEFT$(DU$,1)="N" THEN PRINT:PRINT" Then turn it ON !!!":GOTO 1100
1120 IF DU$="" THEN RETURN
1130 IF LEFT$(DU$,1)<>"Y"THEN 1090
1140 PRINT:FOR I=1 TO 9:PRINT:NEXT
1150 PRINT"               WAIT - LOADING ..."
1160 LPRINT "L":FOR I=1 TO 250:NEXT I:LPRINT CHR$(13)
1180 OPEN "I",1,F$
1190 LINE INPUT #1,A
1200 LPRINT A
1210 PRINT A
1220 IF EOF(1) THEN CLOSE:GOTO 1235
1230 GOTO 1190
1235 LPRINT CHR$(26)
1236 FOR I=1 TO 200:NEXT I
1240 LPRINT "A":FOR I=1 TO 250:NEXT I:LPRINT CHR$(13)
1250 PRINT:PRINT:PRINT
1260 PRINT"On yellow light :  Programmer is busy - Don't"
1270 PRINT"                   Touch anything !!!"
1280 PRINT
1290 PRINT"On green light  :  Insert your Pal and press"
1300 PRINT"                   <PROG> on the programmer":PRINT
1310 PRINT"On red light    :  Assembling error - Your Pal"
1320 PRINT"                   Design Spec is wrong."
1330 FOR I=1 TO 10:PRINT:NEXT
1340 GOTO 1730
```

```
1360 REM **************************************************************
1370 REM ***             X-PLOT FUNCTION                        ***
1380 REM **************************************************************
1390 A1="               11 1111 1111 2222 2222 2233"
1400 A2="   0123 4567 8901 2345 6789 0123 4567 8901"
1410 A3="X = FUSE INTACT   (L,N,0)         - = FUSE BLOWN     (H,P,1)"
1415 A4="o = PHANTOM FUSE  (L,N,0)         O = PHANTOM FUSE   (H,P,1)"
1420 IF TY<8 THEN 1450
1430 A1=A1+" 3333 3333"
1440 A2=A2+" 2345 6789"
1450 GOSUB 3120
1460 C=0
1470 A(0)="X":A(1)="-":A(2)="o":A(3)="O"
1480 IF X$="" THEN 1500
1490 PRINT #1," ":PRINT #1,A1:PRINT #1,A2
1500 PRINT:PRINT A1:PRINT A2
1510 IF (S=19 AND C>63) OR (S=23 AND C>79) THEN 1660
1520 IF S$="X" THEN 1540
1530 IF (F(0,C)=0 AND F(1,C)=0) OR F(0,C)>1 THEN C=C+1:GOTO 1510
1540 GOSUB 1760
1550 IF C/8<>INT(C/8) THEN 1590
1560 IF X$="" THEN 1580
1570 PRINT #1," "
1580 PRINT
1590 IF X$="" THEN 1620
1600 PRINT #1, USING "##";C;
1610 PRINT #1, A
1620 PRINT USING "##";C;
1630 PRINT A
1640 C=C+1
1650 GOTO 1510
1660 IF X$="" THEN 1700
1670 PRINT #1,"":PRINT #1,A3
1672 IF S$="X" THEN PRINT #1,A4
1675 PRINT #1,""
1680 PRINT #1,"NUMBER OF FUSES BLOWN =";NB
1690 PRINT #1," "
1700 PRINT:PRINT A3
1702 IF S$="X" THEN PRINT A4
1705 PRINT
1710 PRINT "NUMBER OF FUSES BLOWN =";NB
1720 PRINT
1730 PRINT "Press any key to continue..."
1740 DUMMY$=INKEY$:IF DUMMY$="" THEN 1740
1750 RETURN
1760 A=" "
1770 IF S$="B" THEN A(2)=" ":A(3)=" "
1780 FOR I=0 TO XM STEP 4
1790   A=A+A(F(I,C))+A(F(I+1,C))+A(F(I+2,C))+A(F(I+3,C))+" "
1800 NEXT I
1810 RETURN
1820 REM ***********************************************************
1830 REM ***     JEDEC FORMAT                                  ***
1840 REM ***********************************************************
1850 PRINT:PRINT"Press 1 for DATA I/O"
1860   PRINT"       2 for File or"
1870 PRINT"<return> for none of both.";
1880 DU$=INKEY$:PRINT DU$;:IF DU$="" THEN 1880
1885 IF DU$=CHR$(13) THEN X$="":GOTO 2000
1890 IF DU$="2" THEN GOSUB 3120:GOTO 2010
1900 IF DU$<>"1" THEN PRINT CHR$(8);:GOTO 1880
1905 PRINT:PRINT:PRINT:PRINT
1910 PRINT"Power on Data I/O."
1920 PRINT:PRINT"For Information about how to setup the Data I/O"
1930 PRINT"to Receive-mode, look in the PALASM manual under"
1940 PRINT"Appendix B.":PRINT
1950 PRINT"Press <return> when ready..."
1960 PRINT:PRINT"After this press <RETURN> on the computer."
1970 S$=INKEY$:IF S$="" THEN 1970
1980 PRINT:PRINT"Please Wait !!!"
1990 PRINT:PRINT"You are now down-loading data to the Data I/O"
2000 GOSUB 3160
2010 SCHK=0:A(0)="0":A(1)="1"
2020 IF FCODE>9 THEN 2050
2030 A=CHR$(2)+"*D220"+RIGHT$(STR$(FCODE),1)+"*F0*"
```

continued

```
2040 GOTO 2060
2050 A=CHR$(2)+"*D22"+RIGHT$(STR$(FCODE),2)+"*F0*"
2060 GOSUB 3020
2070 LINENR=0
2080 J=S-12
2090 IF J<0 THEN 2810
2100 C=N(J)-N1(J)+1
2110 IF N(J)=0 THEN J=J-1:GOTO 2090
2120 IF F(0,C)=1 OR F(1,C)=1 THEN 2220
2130 IF TY=4 OR TY=5 OR TY=6 OR TY=7 OR TY=9 THEN LINENR=LINENR+32
2140 IF TY=11 OR TY=13 OR TY=14 OR TY=15 OR TY=16 THEN LINENR=LINENR+40
2150 IF TY=1 THEN LINENR=LINENR+20
2160 IF TY=2 OR TY=12 THEN LINENR=LINENR+24
2170 IF TY=3 OR TY=8 THEN LINENR=LINENR+28
2180 IF TY=10 THEN LINENR=LINENR+36
2190 C=C+1
2200 IF C=N(J)+1 THEN J=J-1:GOTO 2090
2210 GOTO 2120
2220 I=0
2230 IF TY<>1 THEN 2300
2240 T=1:GOSUB 2750
2250 T=3:GOSUB 2780
2260 T=1:GOSUB 2750
2270 GOSUB 2980
2280 LINENR=LINENR+20
2290 GOTO 2190
2300 IF TY<>2 THEN 2370
2310 T=2:GOSUB 2750
2320 T=2:GOSUB 2780
2330 T=2:GOSUB 2750
2340 GOSUB 2980
2350 LINENR=LINENR+24
2360 GOTO 2190
2370 IF TY<>3 THEN 2440
2380 T=3:GOSUB 2750
2390 T=1:GOSUB 2780
2400 T=3:GOSUB 2750
2410 GOSUB 2980
2420 LINENR=LINENR+28
2430 GOTO 2190
2440 IF TY<>8 THEN 2490
2450 T=2:GOSUB 2750
2460 T=3:GOSUB 2780
2470 T=2:GOSUB 2750
2480 GOTO 2410
2490 IF TY<>9 THEN 2560
2500 T=3:GOSUB 2750
2510 T=2:GOSUB 2780
2520 T=3:GOSUB 2750
2530 GOSUB 2980
2540 LINENR=LINENR+32
2550 GOTO 2190
2560 IF TY<>10 THEN 2620
2570 T=4:GOSUB 2750
2580 T=1:GOSUB 2780
2590 T=4:GOSUB 2750
2595 GOSUB 2980
2600 LINENR=LINENR+36
2610 GOTO 2190
2620 IF TY<>12 THEN 2670
2630 T=1:GOSUB 2750
2640 T=4:GOSUB 2780
2650 T=1:GOSUB 2750
2660 GOTO 2340
2670 GOSUB 2710
2680 GOSUB 2980
2690 LINENR=LINENR+XM+1
2700 GOTO 2190
2710 FOR I=0 TO XM STEP 4
2720   A=A+A(F(I,C))+A(F(I+1,C))+A(F(I+2,C))+A(F(I+3,C))+" "
2730 NEXT I
2740 RETURN
2750 A=A+A(F(I,C))+A(F(I+1,C))+A(F(I+2,C))+A(F(I+3,C))+" "
2760 I=I+4:T=T-1
2770 IF T<1 THEN RETURN ELSE 2750
2780 A=A+A(F(I,C))+A(F(I+1,C))+A(F(I+4,C))+A(F(I+5,C))+" "
```

```
2790 I=I+8:T=T-1
2800 IF T<1 THEN RETURN ELSE 2780
2810 SCHK=SCHK+3
2820 IF SCHK>65535! THEN SCHK=SCHK-65535!
2830 IF DU$="1" THEN LPRINT CHR$(3)+HEX$(SCHK)
2840 IF X$="" THEN 2860
2850 PRINT #1,CHR$(3)+HEX$(SCHK)
2860 PRINT CHR$(3)+HEX$(SCHK)
2870 IF DU$<>"1" THEN 1720
2875 PRINT:PRINT:PRINT:PRINT
2880 PRINT"Now the Data I/O should display a 4 digit number."
2890 PRINT"If not refer to the Error messages in Appendix B in"
2900 PRINT"the PALASM manual.":PRINT
2910 PRINT"Now insert the blank PAL device in the socket below"
2920 PRINT"the red light.":PRINT
2930 PRINT"Refer to Appendix B in the PALASM manual for information"
2940 PRINT"on the programming sequence on the Data I/O"
2970 GOTO 1720
2980 IF LINENR<10 THEN A="L000"+RIGHT$(STR$(LINENR),1)+" "+A+"*":GOTO 3020
2990 IF LINENR<100 THEN A="L00"+RIGHT$(STR$(LINENR),2)+" "+A+"*":GOTO 3020
3000 IF LINENR<1000 THEN A="L0"+RIGHT$(STR$(LINENR),3)+" "+A+"*":GOTO 3020
3010 IF LINENR>999 THEN A="L"+RIGHT$(STR$(LINENR),4)+" "+A+"*"
3020 A=A+CHR$(13)+CHR$(10)
3030 IF DU$="1" THEN LPRINT A;
3040 IF X$="" THEN 3060
3050 PRINT #1,A;
3060 PRINT A;
3070 FOR I=1 TO LEN(A)
3080   SCHK=SCHK+ASC(MID$(A,I,1))
3090 NEXT I
3100 A=""
3110 RETURN
3120 PRINT:PRINT"Enter filename for output ( Return for none ) "
3130 LINE INPUT X$
3140 IF X$="" THEN 3160
3150 OPEN "O",1,X$
3160 OPEN "I",2,F$
3162 IF X$="" THEN 3165
3163 PRINT #1,"(c) Copyright 1983 Monolithic Memories Inc. All Rights
Reserved."
3165 PRINT"(c) Copyright 1983 Monolithic Memories Inc. All Rights Reserved."
3170 FOR I=1 TO 4
3180   LINE INPUT #2,A
3190   IF X$="" THEN 3210
3200   PRINT #1,A
3210   PRINT A
3220 NEXT I
3230 CLOSE 2
3240 RETURN
3250 REM ******************************************************
3260 REM ***            ECHO                                ***
3270 REM ******************************************************
3280 OPEN "I",1,F$
3282 I=0
3284 LINE INPUT #1,A
3286 PRINT A
3288 I=I+1
3290 IF EOF(1) THEN 1720
3292 IF I<22 THEN 3284
3294 GOSUB 1730
3296 GOTO 3282
3340 REM ****************************************************
3350 REM ***          PINOUT                              ***
3360 REM ****************************************************
3370 GOSUB 3120
3380 IF X$="" THEN 3420
3390 PRINT #1,""
3400 PRINT #1,TAB(20);"**************   **************"
3410 PRINT #1,TAB(20);"*             ***             *"
3420 PRINT
3430 PRINT TAB(20);"**************   **************"
3440 PRINT TAB(20);"*             ***             *"
3450 FOR I=1 TO (S+1)/2
3460 A=""
3470 GOSUB 3810
```

continued

```
3480 A=SPACE$(13-LEN(P(I)))+P(I)+"   *"
3490 IF I>9 THEN 3520
3500 A=A+" "+RIGHT$(STR$(I),1)+"*"
3510 GOTO 3530
3520 A=A+RIGHT$(STR$(I),2)+"*"
3530 IF I>1 THEN 3560
3540 A=A+SPACE$(12)+"P A L"+SPACE$(12)
3550 GOTO 3570
3560 A=A+SPACE$(29)
3570 A=A+"*"+RIGHT$(STR$(((S+2)-I)),2)+"*   "+P((S+2)-I)
3580 IF X$="" THEN 3600
3590 PRINT #1,A
3600 PRINT A
3610 GOSUB 3810
3620 IF I>1 THEN GOSUB 3760:GOTO 3710
3630 A=SPACE$(19)+"*"+SPACE$(16-(LEN(TYPE$)+1))
3640 FOR J=1 TO LEN(TYPE$)
3650   A=A+MID$(TYPE$,J,1)+" "
3660 NEXT
3670 A=A+SPACE$(49-LEN(A))+"*"
3680 IF X$="" THEN 3700
3690 PRINT #1,A
3700 PRINT A
3710 NEXT I
3720 IF X$="" THEN 3740
3730 PRINT #1,TAB(20);"******************************"
3740 PRINT TAB(20);"******************************"
3750 GOTO 1720
3760 REM  *..........*
3770 IF X$="" THEN 3790
3780 PRINT #1,TAB(20);"*                            *"
3790 PRINT TAB(20);"*                            *"
3800 RETURN
3810 REM  ****...................****
3820 IF X$="" THEN 3840
3830 PRINT #1,TAB(17);"****                             ****"
3840 PRINT TAB(17);"****                             ****"
3850 RETURN
```

PALTABLE.DAT
Contributed by: Robert A. Freedman
"Getting Started with PALs," by Robert A. Freedman. January, page 223.

```
10L8,1,31,63,19,13,3,1,5,9,13,17,21,25,29,0,31,0,0,0,0,0,0,0,0,57,2,49,2,41,2,
33,2,25,2,17,2,9,2,1,2,4,4,4,4,4,4,4,4
10H8,1,31,63,19,18,3,1,5,9,13,17,21,25,29,0,31,0,0,0,0,0,0,0,0,57,2,49,2,41,2,
33,2,25,2,17,2,9,2,1,2,4,4,4,4,4,4,4,4
12H6,2,31,63,19,19,3,1,5,9,13,17,21,25,29,0,31,27,0,0,0,0,0,0,7,0,0,49,4,41,2,
33,2,25,2,17,2,9,4,0,0,2,6,5,5,5,5,6,2
12L6,2,31,63,19,14,3,1,5,9,13,17,21,25,29,0,31,27,0,0,0,0,0,0,7,0,0,49,4,41,2,
33,2,25,2,17,2,9,4,0,0,2,6,5,5,5,5,6,2
14H4,3,31,63,19,20,3,1,5,9,13,17,21,25,29,0,31,27,23,0,0,0,0,11,7,0,0,0,0,41,4
,33,4,25,4,17,4,0,0,0,0,2,2,7,7,7,7,2,2
14L4,3,31,63,19,15,3,1,5,9,13,17,21,25,29,0,31,27,23,0,0,0,0,11,7,0,0,0,0,41,4
,33,4,25,4,17,4,0,0,0,0,2,2,7,7,7,7,2,2
16H2,4,31,63,19,22,3,1,5,9,13,17,21,25,29,0,31,27,23,19,0,0,15,11,7,0,0,0,0,0,
0,33,8,25,8,0,0,0,0,0,0,2,2,2,1,1,2,2,2
16L2,4,31,63,19,16,3,1,5,9,13,17,21,25,29,0,31,27,23,19,0,0,15,11,7,0,0,0,0,0,
0,33,8,25,8,0,0,0,0,0,0,2,2,2,1,1,2,2,2
16C1,4,31,63,19,21,3,1,5,9,13,17,21,25,29,0,31,27,23,19,0,0,15,11,7,0,0,0,0,0,
0,25,16,25,16,0,0,0,0,0,0,2,2,2,1,1,3,3,3
16L8,5,31,63,19,17,3,1,5,9,13,17,21,25,29,0,31,0,27,23,19,15,11,7,0,-57,8,-
49,8,-41,8,-33,8,-25,8,-17,8,-9,8,-1,8,1,1,1,1,1,1,1,1
16R4,6,31,63,19,24,0,1,5,9,13,17,21,25,29,0,0,31,27,23,19,15,11,7,3,-57,8,-
49,8,141,8,133,8,125,8,117,8,-9,8,-1,8,1,1,1,1,1,1,1,1
16R6,6,31,63,19,24,0,1,5,9,13,17,21,25,29,0,0,31,27,23,19,15,11,7,3,-
57,8,149,8,141,8,133,8,125,8,117,8,109,8,-1,8,1,1,1,1,1,1,1,1
16R8,6,31,63,19,24,0,1,5,9,13,17,21,25,29,0,0,31,27,23,19,15,11,7,3,157,8,149,
8,141,8,133,8,125,8,117,8,109,8,101,8,1,1,1,1,1,1,1,1
16A4,7,31,63,19,24,0,1,5,8,12,16,20,25,29,0,0,31,27,0,0,0,0,7,3,-57,8,-
49,8,141,8,133,8,125,8,117,8,-9,8,-1,8,1,1,1,1,1,1,1,1
16X4,7,31,63,19,24,0,1,5,8,12,16,20,25,29,0,0,31,27,0,0,0,0,7,3,-57,8,-
49,8,141,8,133,8,125,8,117,8,-9,8,-1,8,1,1,1,1,1,1,1,1
```

```
14L8,8,39,79,23,2,3,1,5,9,13,17,21,25,29,33,37,0,39,35,0,0,0,0,0,0,0,7,0,0,0
,0,0,0,65,4,57,2,49,2,41,2,33,2,25,2,17,2,9,4,0,0,3,6,5,5,5,5,5,5,6,3
16L6,9,39,79,23,3,3,1,5,9,13,17,21,25,29,33,37,0,39,35,31,0,0,0,0,0,0,11,7,0,0
,0,0,0,0,0,0,57,4,49,4,41,2,33,2,25,4,17,4,0,0,0,0,3,3,7,7,8,8,7,7,3,3
18L4,10,39,79,23,4,3,1,5,9,13,17,21,25,29,33,37,0,39,35,31,27,0,0,0,0,15,11,7,
0,0,0,0,0,0,0,0,0,49,6,41,4,33,4,25,6,0,0,0,0,0,0,3,3,3,9,10,10,9,3,3,3
20L2,11,39,79,23,5,3,1,5,9,13,17,21,25,29,33,37,0,39,35,31,27,23,0,0,19,15,11,
7,0,0,0,0,0,0,0,0,0,0,41,8,33,8,0,0,0,0,0,0,0,3,3,3,3,1,1,3,3,3,3
20C1,11,39,79,23,12,3,1,5,9,13,17,21,25,29,33,37,0,39,35,31,27,23,0,0,19,15,11
,7,0,0,0,0,0,0,0,0,0,0,33,16,33,16,0,0,0,0,0,0,0,2,2,2,2,1,1,3,3,3,3
12L10,12,39,79,23,1,3,1,5,9,13,17,21,25,29,33,37,0,39,0,0,0,0,0,0,0,0,0,0,0,
0,0,73,2,65,2,57,2,49,2,41,2,33,2,25,2,17,2,9,2,1,2,4,4,4,4,4,4,4,4,4,4
20L10,13,39,79,23,6,3,1,5,9,13,17,21,25,29,33,37,0,39,0,35,31,27,23,19,15,11,7
,0,0,0,0,0,-73,4,-65,4,-57,4,-49,4,-41,4,-33,4,-25,4,-17,4,-9,4,-
1,4,11,11,11,11,11,11,11,11,11,11
20L8,14,39,79,23,26,3,1,5,9,13,17,21,25,29,33,37,0,39,35,0,31,27,23,19,15,11,0
,7,0,0,0,0,0,-65,8,-57,8,-49,8,-41,8,-33,8,-25,8,-17,8,-
9,8,0,0,3,1,1,1,1,1,1,1,1,3
20X10,15,39,79,23,23,0,1,5,9,13,17,21,25,29,33,37,0,0,39,35,31,27,23,19,15,11,
7,3,0,0,0,0,173,4,165,4,157,4,149,4,141,4,133,4,125,4,117,4,109,4,101,4,11,11,
11,11,11,11,11,11,11,11
20X8,15,39,79,23,23,0,1,5,9,13,17,21,25,29,33,37,0,0,39,35,31,27,23,19,15,11,7
,3,0,0,0,0,-73,4,165,4,157,4,149,4,141,4,133,4,125,4,117,4,109,4,-
1,4,11,11,11,11,11,11,11,11,11,11
20X4,15,39,79,23,23,0,1,5,9,13,17,21,25,29,33,37,0,0,39,35,31,27,23,19,15,11,7
,3,0,0,0,0,-73,4,-65,4,-57,4,149,4,141,4,133,4,125,4,-17,4,-9,4,-
1,4,11,11,11,11,11,11,11,11,11,11
20R8,16,39,79,23,27,0,1,5,9,13,17,21,25,29,33,37,0,0,39,35,31,27,23,19,15,11,7
,3,0,0,0,0,0,0,165,8,157,8,149,8,141,8,133,8,125,8,117,8,109,8,0,0,3,1,1,1,1,1
,1,1,1,3
20R6,16,39,79,23,27,0,1,5,9,13,17,21,25,29,33,37,0,0,39,35,31,27,23,19,15,11,7
,3,0,0,0,0,0,0,-65,8,157,8,149,8,141,8,133,8,125,8,117,8,-
9,8,0,0,3,1,1,1,1,1,1,1,1,3
20R4,16,39,79,23,27,0,1,5,9,13,17,21,25,29,33,37,0,0,39,35,31,27,23,19,15,11,7
,3,0,0,0,0,0,0,-65,8,-57,8,149,8,141,8,133,8,125,8,-17,8,-
9,8,0,0,3,1,1,1,1,1,1,1,1,3
```

LINKLIST.PAS
Contributed by: Antonio Fernandes
Programming Insight: "Dynamic Memory Allocation," by Antonio Fernandes.
January, page 169.

```
(*******************************************************************)
(*                          LINKLIST.PAS                           *)
(*                                                                 *)
(* This program maintains an ordered linked list of strings.  It is *)
(* designed for an Apple IIe or an Apple II+ with an 80-column card. *)
(* If you're using Pascal with 40 columns, all that has to be changed *)
(* is the number 40 in the GOTOXY calls.  A printer is also assumed. *)
(* If you have none online, then one procedure call must be removed, *)
(* and it is marked as such in the program.                        *)
(*                                                                 *)
(* The program commands are as follows:                            *)
(*      A)dd - adds string to list                                 *)
(*      D)elete - deletes string from list                         *)
(*      B)lank - destroys list                                     *)
(*      P)rint - dumps list to printer                             *)
(*      E)nd - terminates program                                  *)
(* Please note that on an Apple IIe the Caps Lock button must be   *)
(* depressed for the program to accept these commands.             *)
(*******************************************************************)

PROGRAM MANAGER;

{ WARNING: shuts off range checking }
(*$R-*)

TYPE
        COMCHARS=SET OF CHAR;

        ST1=STRING[15];
        ST2=STRING[6];
```

continued

```
        LISTPOINT=^NODE;
        NODE=RECORD
                NAME:ST1;
                LINK:LISTPOINT;
             END;

VAR
        LIST      : LISTPOINT;  { linked list head }
        TARGET    : ST1;        { string to be manipulated }
        COMMAND   : CHAR;       { input operation to list }
        PROPCOMS  : COMCHARS;   { set of proper commands }
        HEAP      : ^INTEGER;   { holds heap marker }

(*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*)
(*                                                                *)
(*                    PROCEDURES AND FUNCTIONS                    *)
(*                                                                *)
(*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*)

        (**********************************************)
        (*                                            *)
        (*                  Previous                  *)
        (*                                            *)
        (**********************************************)

{ this procedure returns a pointer to the node before the target node or nil }

FUNCTION PREVIOUS(LIST:LISTPOINT; TARGET:ST1):LISTPOINT;

VAR
        CURRENT:LISTPOINT; { current position pointer }

BEGIN
        { CURRENT is initialized to beginning of list }
        CURRENT:=LIST;

        { move node pointer until target or end of list is encountered }
        WHILE (CURRENT^.LINK^.NAME<TARGET) AND (CURRENT^.LINK<>NIL)
        DO BEGIN
             CURRENT:=CURRENT^.LINK
           END;

        PREVIOUS:=CURRENT
END;

        (**********************************************)
        (*                                            *)
        (*                    Add                     *)
        (*                                            *)
        (**********************************************)

        { adds a node to the list in between two nodes }

PROCEDURE ADD(VAR PREV:LISTPOINT);

VAR
        TEMP:LISTPOINT;

BEGIN
        TEMP:=PREV^.LINK;
        NEW(PREV^.LINK);
        PREV^.LINK^.NAME:=TARGET;
        PREV^.LINK^.LINK:=TEMP
END;

        (**********************************************)
        (*                                            *)
        (*                  Insert                    *)
        (*                                            *)
        (**********************************************)

                { insert new node in list }

PROCEDURE INSERT(LIST:LISTPOINT;TARGET:ST1);
```

```
VAR
        PREV:LISTPOINT;

BEGIN
        PREV:=PREVIOUS(LIST,TARGET);

        { check to see if element is already in list }
        IF PREV^.LINK^.NAME=TARGET
          THEN
            BEGIN
                GOTOXY(40,10);
                WRITELN('Element in list alread');
                WRITE(CHR(7));WRITE(CHR(7))
            END
          ELSE
            ADD(PREV)
END;

        (************************************************)
        (*                                              *)
        (*                  Subtract                    *)
        (*                                              *)
        (************************************************)

             { remove a node in between two others }

PROCEDURE SUBTRACT(VAR PREV:LISTPOINT);

BEGIN
        PREV^.LINK:=PREV^.LINK^.LINK
END;

        (************************************************)
        (*                                              *)
        (*                  Delete                      *)
        (*                                              *)
        (************************************************)

                { delete a node from list }

PROCEDURE DELETE(LIST:LISTPOINT;TARGET:ST1);

VAR
        PREV:LISTPOINT;

BEGIN
        PREV:=PREVIOUS(LIST,TARGET);

        { check to see if target is in list }
        IF (PREV^.LINK=NIL) OR (PREV^.LINK^.NAME<>TARGET)
          THEN
            BEGIN{ then }
                GOTOXY(40,10);
                WRITELN('Target not found !!!');
                WRITE(CHR(7));WRITE(CHR(7))
            END{ then }
          ELSE
            SUBTRACT(PREV)
END;

        (************************************************)
        (*                                              *)
        (*                  Show Mem                    *)
        (*                                              *)
        (************************************************)

                    { show memory left }

PROCEDURE SHOW_MEM;

BEGIN
  { uses the built-in function MEMAVAIL to return number of works left }

        GOTOXY(40,20);
```

continued

```
        { check to see if there is a reasonable amount of memory left }
        IF MEMAVAIL>100
          THEN
            WRITE('There are ',MEMAVAIL:5,' words of memory left')
          ELSE
            BEGIN
                WRITELN(':NEARING END OF MEMORY !!');
                WRITE(CHR(7));WRITE(CHR(7))
            END
END;

        (**********************************************)
        (*                                            *)
        (*                 Print List                 *)
        (*                                            *)
        (**********************************************)

          { send contents of list to device specified }

PROCEDURE PRINT_LIST(LIST:LISTPOINT;DEVICE:ST1);

VAR
        CURRENT:LISTPOINT;
        OUT:TEXT; { variable representing output file }
BEGIN

        { set up device communication }
        REWRITE(OUT,DEVICE);
        PAGE(OUT);

        CURRENT:=LIST;

        { send information to device }
        WRITELN(OUT,'Current elements in the list are:');
        WRITELN(OUT,'------- -------- -- --- ---- ---');

        WHILE CURRENT^.LINK<>NIL
        DO BEGIN
             WRITELN(OUT,CURRENT^.LINK^.NAME);
             CURRENT:=CURRENT^.LINK
           END;

        SHOW_MEM
END; { print_list }

        (**********************************************)
        (*                                            *)
        (*                  Get Name                  *)
        (*                                            *)
        (**********************************************)

                 { get string to be manipulated }

PROCEDURE GET_NAME(VAR TARGET:ST1;PROC:ST2);

BEGIN
        GOTOXY(40,2);
        WRITELN('Which string do you wish to ',PROC);
        GOTOXY(40,3);
        WRITE('-----> ');
        READ(TARGET)
END;

        (**********************************************)
        (*                                            *)
        (*                  Kill List                 *)
        (*                                            *)
        (**********************************************)

                  { destroy contents of list }

PROCEDURE KILL_LIST(LIST:LISTPOINT);

BEGIN
        RELEASE(HEAP);
        { tie up only link remaining after heap is destroyed }
        LIST^.LINK:=NIL;

        PAGE(OUTPUT);
        WRITELN('List is now empty.');
        SHOW_MEM
END;

        (************************************************)
        (*                                              *)
        (*                  Initialize                  *)
        (*                                              *)
        (************************************************)

                        { create first node }

PROCEDURE INITIALIZE(VAR LIST:LISTPOINT);

BEGIN
        { create list head }
        NEW(LIST);
        LIST^.LINK:=NIL;

        { set heap pointer }
        MARK(HEAP)
END;

(*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*)
(*                                                              *)
(*                      MAIN PROGRAM                            *)
(*                                                              *)
(*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*)

BEGIN{ main }
        INITIALIZE(LIST);

        { set of proper inputs }
        PROPCOMS:=['A','D','P','B','E'];

        WHILE COMMAND<>'E'
        DO BEGIN{ while }

                GOTOXY(40,0);
                WRITE('A(dd  B(lank  D(elete  P(rint  E(nd ');
                READ(COMMAND);
                IF (COMMAND IN PROPCOMS)
                  THEN
                  CASE COMMAND OF
                    'A':BEGIN
                          GET_NAME(TARGET,'add');
                          INSERT(LIST,TARGET);
                          PRINT_LIST(LIST,'CONSOLE:')
                        END;

                    'D':BEGIN
                          GET_NAME(TARGET,'delete');
                          DELETE(LIST,TARGET);
                          PRINT_LIST(LIST,'CONSOLE:')
                        END;

                    { if you have no printer delete PRINT_LIST call
                      with PRINTER: as a parameter }

                    'P':BEGIN
                          PRINT_LIST(LIST,'PRINTER:');
                          PRINT_LIST(LIST,'CONSOLE:')
                        END;

                   'B':KILL_LIST(LIST)
               END{ case }

       END{ while }

END.{ main }
```

continued

LISTINGS.DOC
Contributed by: Paul A. Sand
"The Stride 440," by Paul A. Sand. January, page 295.

```
program filewrite;
{ UCSD Pascal program to write a 64K data file }
{ in 512 chunks of 128 bytes each              }

const
        CHUNK_SIZE = 128;                { size of chunks in bytes }
        N_CHUNKS = 512;          { number of chunks to write }

type
        chunk_array = packed array [1..CHUNK_SIZE] of char;
        chunk_file = file of chunk_array;

var
        chunk : chunk_array;     { one chunk }
        cf : chunk_file;                 { file variable for data file }
        i : integer;                     { loop control variable }

begin { filewrite }
        for i := 1 to CHUNK_SIZE do
                chunk[i] := chr(ord('1') + (i - 1) mod 8);
        rewrite(cf, 'TEST');
        for i := 1 to N_CHUNKS do begin
                cf^ := chunk;
                put(cf)
        end;
        close(cf, LOCK)
end.
```

Listing 1
UCSD Pascal 64Kbyte Fileπwriting Benchmark

```
program fileread;
{ UCSD Pascal program to read a 64K data file }
{ in 512 chunks of 128 bytes each             }

const
        CHUNK_SIZE = 128;                { size of chunks in bytes }
        N_CHUNKS = 512;          { number of chunks to read }

type
        chunk_array = packed array [1..CHUNK_SIZE] of char;
        chunk_file = file of chunk_array;

var
        chunk : chunk_array;     { one chunk }
        cf : chunk_file;                 { file variable for data file }
        i : integer;                     { loop control variable }

begin { fileread }
        reset(cf, 'TEST');
        chunk := cf^;
        for i := 2 to N_CHUNKS do begin
                get(cf);
                chunk := cf^;
        end;
        close(cf)
end.
```

Listing 2
UCSD Pascal 64Kbyte Fileπreading Benchmark

```
program calculations;
{ UCSD Pascal program to perform series of real }
{ multiplications and divisions }

const
        MAX = 5000;     { number of repetitions }
        var
                a, b, c : real; { used in calculations }
                i : integer;                    { loop control variable }

        begin { calculations }
                a := 2.71828;
                b := 3.14159;
                c := 1.0;
                for i := 1 to MAX do begin
                        c := c * a;
                        c := c * b;
                        c := c / a;
                        c := c / b
                end;
                writeln('Error : ', c - 1.0)
        end.
```

Listing 3
UCSD Pascal Calculation Benchmark

```
        program sieve;
        { UCSD Pascal Sieve of Eratosthenes Benchmark }

        const
                SIZE = 7000;            { size of array for standard benchmark }

        var
                flags : array [0..SIZE] of Boolean;
                i, prime, k, count, iter: integer;

        begin { sieve }
                writeln('10 iterations');
                for iter := 1 to 10 do begin
                        count := 0;
                        for i := 0 to SIZE do
                                flags[i] := TRUE;
                        for i := 0 to SIZE do
                                if flags[i] then begin
                                        prime := i + i + 3;
                                        k := i + prime;
                                        while k <= SIZE do begin
                                                flags[k] := FALSE;
                                                k := k + prime
                                        end;
                                        count := count + 1
                                end
                end;
                writeln(count, ' primes')
        end.
```

Listing 4
UCSD Pascal Sieve of Eratosthenes Benchmark

```
MODULE filewrite;

FROM Files IMPORT
    FILE, FileState, Create, Close, WriteBytes;

FROM SYSTEM IMPORT
    ADR;

CONST
    CHUNKSIZE = 128;   (* size of chunks in bytes *)
    NCHUNKS = 512;     (* number of chunks to write *)

TYPE
    chunkarray = ARRAY [1..CHUNKSIZE] OF CHAR;

VAR
        chunk : chunkarray;             (* one chunk *)
        cf : FILE;                      (* chunk file variable *)
        i : CARDINAL;                   (* loop control variable *)
        junk : CARDINAL;                (* status from WriteBytes *)
        fs : FileState;                 (* status from Create/Close *)
        name : ARRAY [0..30] OF CHAR;   (* file name *)
```

continued

```
BEGIN
    FOR i := 1 TO CHUNKSIZE DO
        chunk[i] := CHR(ORD('1') + (i - 1) MOD 8)
    END;
    name := "TEST.DATA";
    fs := Create(cf, name);
    FOR i := 1 TO NCHUNKS DO
        junk := WriteBytes(cf, ADR(chunk), CHUNKSIZE)
    END;
    fs := Close(cf)
END filewrite.
```

Listing 5
Modulaπ2 64Kbyte File Writing Benchmark

```
MODULE fileread;

FROM Files IMPORT
    FILE, FileState, Open, Close, ReadBytes;

FROM SYSTEM IMPORT
    ADR;

CONST
    CHUNKSIZE = 128;    (* size of chunks in bytes *)
    NCHUNKS = 512;      (* number of chunks to write *)

TYPE
    chunkarray = ARRAY [1..CHUNKSIZE] OF CHAR;

VAR
        chunk : chunkarray;                         (* one chunk *)
        cf : FILE;                                          (* chunk file variable *)
        i : CARDINAL;                               (* loop control variable *)
        junk : CARDINAL;                                    (* status from ReadBytes *)
        fs : FileState;                             (* status from Open & Close *)
        name : ARRAY [0..30] OF CHAR;   (* file name *)

BEGIN
    name := "TEST.DATA";
    fs := Open(cf, name);
    FOR i := 1 TO NCHUNKS DO
        junk := ReadBytes(cf, ADR(chunk), CHUNKSIZE)
    END;
    fs := Close(cf);
END fileread.
```

Listing 6
Modulaπ2 64Kbyte File Reading Benchmark

```
MODULE calculations;
(* Modulaπ2 program to perform a series of real *)
(* multiplications and divisions *)

FROM RealInOut IMPORT
    WriteReal;

FROM InOut IMPORT
    WriteString, WriteLn;

CONST
    MAX = 5000;         (* number of iterations *)

VAR
    a, b, c : REAL;     (* used in calculations *)
    i : CARDINAL;       (* loop control variable *)

BEGIN
    a := 2.71828;
    b := 3.14159;
    c := 1.0;
    FOR i := 1 TO MAX DO
        c := c * a;
        c := c * b;
        c := c / a;
        c := c / b
    END;
    WriteString('Error = ');
    WriteReal(c - 1.0, 10);
    WriteLn
END calculations.
```

Listing 7
Modulaπ2 Real Calculations Benchmark

```
MODULE sieve;

FROM InOut IMPORT
    WriteLn, WriteString, WriteCard;

CONST
    SIZE = 7000;

VAR
    flags : ARRAY [0..SIZE] OF BOOLEAN;
    i, prime, k, count, iter : CARDINAL;

BEGIN
    WriteString('10 iterations');
    FOR iter := 1 TO 10 DO
        count := 0;
        FOR i := 0 TO SIZE DO
            flags[i] := TRUE
        END;
        FOR i := 0 TO SIZE DO
            IF flags[i] THEN
                prime := i + i + 3;
                k := i + prime;
                WHILE k <= SIZE DO
                    flags[k] := FALSE;
                    k := k + prime
                END;
                INC(count)
            END
        END
    END;
    WriteCard(count, 1);
    WriteString(' primes');
    WriteLn
END sieve.
```

Listing 8
Modulaπ2 Sieve of Eratosthenes Benchmark
Program Benchmarks

README.PAL
Contributed by: Trevor G. Marshall
"PALs Simplify Complex Circuits," by Trevor G. Marshall. January, page 247.

The Fortran-77 source files for PALASM version 1.3 are PALASM.FOR, SIMLT.FOR, and FILENAME.FOR. The files TESTAS1.PAL, TESTAS2.PAL, TSTHOLD1.PAL, TSTHOLD2.PAL, AND TSTHOLD3.PAL are the listings printed in the article, "PALs Make Complex Circuits Simpler."

FILENAME.FOR
Contributed by: Trevor G. Marshall
"PALs Simplify Complex Circuits," by Trevor G. Marshall. January, page 247.

```
      SUBROUTINE GFNAME(NAME,UNIT,EXT)
      LOGICAL NAME(11),EXT
      INTEGER UNIT
      LOGICAL COLON,FNAME(14),DOT
      COLON = ':'
      DOT   = '.'
```

continued

```
C
C        READ A USER DATA FILE NAME
C
C        NAME:   RETURN PARAMETER OF FILE NAME
C        UNIT:   RETURN PARAMETER OF UNIT
C                        0       DEFAULT DRIVE
C                        1       DRIVE A ETC
C        EXT:    INPUT PARAMETER
C                TRUE : EXTENSION REQUIRED
C                FALSE: EXTENSION NOT PERMITTED
C
C
C
         UNIT = 0
  10     IF(.NOT.EXT)WRITE(1,101)
         IF(EXT) WRITE(1,100)
         READ(1,104)FNAME
         IF(FNAME(2).NE.COLON) GOTO 14
         UNIT = FNAME(1) - 'A' + 1
         DO 15 I= 1  , 12
                 FNAME(I)=FNAME(I+2)
  15     CONTINUE
C
C        FIND AND REMOVE '.' IN FILE NAME
C
   14    DO 16 I=1,11
         IF (FNAME(I).EQ.DOT) GO TO 17
   16    CONTINUE
C        NO PERIOD! ILLEGAL FILE NAME?
         IF(.NOT. EXT) GO TO 20
         WRITE(1,105)
         GOTO 10

   17    M1 = I + 3
         DO 33 J = I,M1
   33    FNAME(J) = FNAME(J+1)
         N=11
         DO 18 J=1,3
         M1=I+3 - J
         FNAME(N)=FNAME(M1)
   18    N=N-1
         IF(I.GE.9)GOTO 20
         DO 19 J=I  ,8
   19    FNAME(J)=' '
C
C        FILE NAME IS O.K
C        TRANSFER TO PARAMETER
C
   20    DO 25 I=1,11
   25    NAME(I) = FNAME(I)
         RETURN
  100    FORMAT(' ENTER FILENAME (WITH EXTENSION) ---> ')
  101    FORMAT(' ENTER FILENAME (WITHOUT EXTENSION -> ')
  104    FORMAT(14A1)
  105    FORMAT(' ILLEGAL FILENAME! PLEASE REENTER')
         END
```

PALASM.FOR
Contributed by: Trevor G. Marshall
"PALs Simplify Complex Circuits," by Trevor G. Marshall. January, page 247.

```
C************************************************************************
C
C
C      MAIN PROGRAM
C
      BYTE    IPAL(4),REST(73),PATNUM(80),TITLE(80),COMP(80),
     C        ISYM(8,20),IBUF(8,20)
      BYTE    E,O,T,P,B,H,S,L,N,Q,U,F,C,R,A,
     C        BB,CC,DD,EE,FF,II,NN,OO,PP,RR,SS,TT,UU,
     C        IPAGE,FNAME(11),MYLINE(80),
     C        INOAI,IOT,INOO,CR,LF,IOP,CLRS
      LOGICAL LBLANK,LLEFT,LAND,LOR,LSLASH,LEQUAL,LRIGHT,LXOR,LXNOR,
     C       LFIX,LFIRST,LMATCH,LFUSES(32,64),LPHASE(20),LBUF(20),
     C       LPROD(80),LSAME,LACT,LOPERR,LINP,LPRD,LHEAD
      COMMON  LBLANK,LLEFT,LAND,LOR,LSLASH,LEQUAL,LRIGHT,LXOR,LXNOR
      COMMON /PGE/ IPAGE(80,100)
      COMMON /FTEST/ IFUNCT,IDESC,IEND
      DATA E/'E'/,O/'O'/,T/'T'/,P/'P'/,B/'B'/,H/'H'/,S/'S'/,L/'L'/,
     C     N/'N'/,Q/'Q'/,U/'U'/,F/'F'/,C/'C'/,R/'R'/,A/'A'/
      DATA BB/'B'/,CC/'C'/,DD/'D'/,EE/'E'/,FF/'F'/,II/'I'/,NN/'N'/,
     C     OO/'O'/,PP/'P'/,RR/'R'/,SS/'S'/,TT/'T'/,UU/'U'/
      DATA CR/X'0D'/,LF/X'0A'/,CLRS/X'0C'/

 999  IFUNCT=0
      IDESC=0
      LSAME=.FALSE.
      LACT=.FALSE.
      LOPERR=.FALSE.
      LINP=.FALSE.
      LPRD=.FALSE.
      LHEAD=.FALSE.
C
      WRITE(1,3)CLRS
    3   FORMAT(' ',A1,'  PAL ASSEMBLER  VERSION  3.1    ',/////)
  530   CALL GFNAME(FNAME,INUNIT,.TRUE.)
        CALL OPEN(6,FNAME,INUNIT)
      READ(6,10,END=500) IPAL,INOAI,IOT,INOO,REST,PATNUM,TITLE,COMP
   10 FORMAT(4A1,A1,A1,A1,73A1,/,80A1,/,80A1,/,80A1)
        GOTO 510
 500    WRITE(1,520)
        ENDFILE 6
 520    FORMAT(' FILE DOESN''T EXIST, REENTER',/)
        GOTO 530

C
 510  WRITE(1,511) IPAL,INOAI,IOT,INOO,REST,PATNUM,TITLE,COMP
 511  FORMAT(' '4A1,A1,A1,A1,73A1,/,' ',80A1,/,
     C       ' ',80A1,/,' ',80A1)
      DO 15 J=1,100
          READ(6,11,END=16) MYLINE
   11     FORMAT(80A1)
        WRITE(1,561)MYLINE
 561    FORMAT(' ',80A1)
        DO 560 I = 1,80
        IPAGE(I,J) = ' '
 560    IF(.NOT.((MYLINE(I).EQ.CR).OR.(MYLINE(I).EQ.LF)))
     C     IPAGE(I,J) = MYLINE(I)

      IF(      IFUNCT.EQ.0 .AND.IPAGE(1,J).EQ.FF.AND.
     C    IPAGE(3,J).EQ.NN.AND.IPAGE(5,J).EQ.TT.AND.
     C    IPAGE(7,J).EQ.OO.AND.IPAGE(10,J) .EQ.TT ) IFUNCT=J
      IF(       IDESC.EQ.0 .AND.IPAGE(1,J).EQ.DD.AND.
     C    IPAGE(3,J).EQ.SS.AND.IPAGE(5,J).EQ.RR.AND.
     C    IPAGE(7,J).EQ.PP.AND.IPAGE(10,J) .EQ.OO ) IDESC=J
   15 CONTINUE
   16 IEND=J-1
      CALL INITLZ(INOAI,IOT,INOO,ITYPE,LFUSES,IC,IL,IBLOW,LFIX)
      ILE=IL+1
      IF(ITYPE.NE.0) GO TO 17
          WRITE(1,18) IPAL,INOAI,IOT,INOO
   18     FORMAT(/,' PAL PART TYPE ',4A1,A1,A1,A1,' IS INCORRECT')
          STOP ERROR
   17 DO 20 J=1,20
   20     CALL GETSYM(LPHASE,ISYM,J,IC,IL,LFIX)
          IF(.NOT.(LEQUAL.OR.LLEFT.OR.LAND.OR.LOR.OR.LRIGHT)) GO TO 24
              WRITE(1,23)
   23         FORMAT(/,' LESS THAN 20 PIN NAMES IN PIN LIST')
              STOP ERROR
   24 ILE=IL
   25 CALL GETSYM(LBUF,IBUF,1,IC,IL,LFIX)
   28     IF(.NOT.LEQUAL) GO TO 25
          COUNT=0
          ILL=IL
          CALL MATCH(IMATCH,IBUF,ISYM)
          IF( IMATCH.EQ.0 ) GO TO 100
          IPRD=IMATCH
          LSAME = ( (     LPHASE(IMATCH)).AND.(     LBUF(1)).OR.
```

continued

```
  C               (.NOT.LPHASE(IMATCH)).AND.(.NOT.LBUF(1)) )
       IF( IOT.EQ.H.AND.(.NOT.LSAME) )                  LACT=.TRUE.
       IF( (.NOT.(IOT.EQ.H.OR.IOT.EQ.C)).AND.(LSAME) ) LACT=.TRUE.
       IF( (ITYPE.EQ.1.OR.ITYPE.EQ.5.OR.ITYPE.EQ.6).AND.IOT.NE.A.
  C    AND.(IMATCH.LT.12.OR.IMATCH.GT.19) ) LOPERR=.TRUE.
       IF(  ITYPE.EQ.2.AND.(IMATCH.LT.13.OR.IMATCH.GT.18) )
  C                                          LOPERR=.TRUE.
       IF(  ITYPE.EQ.3.AND.(IMATCH.LT.14.OR.IMATCH.GT.17) )
  C                                          LOPERR=.TRUE.
       IF(  ITYPE.EQ.4.AND.(IMATCH.LT.15.OR.IMATCH.GT.16) )
  C                                          LOPERR=.TRUE.
       IF( (LACT).OR.(LOPERR) ) GO TO 100
       I88PRO=(19-IMATCH)*8 + 1
       IF(IOT.EQ.C) I88PRO=25
       IC=0
30       CALL INCR(IC,IL,LFIX)
         IF( .NOT.(LEQUAL.OR.LLEFT) ) GO TO 30
         LPROD(I88PRO)=.TRUE.
         IF(.NOT.LLEFT) CALL SLIP(LFUSES,I88PRO,INOAI,IOT,INOO,IBLOW)
         DO 70 I8PRO=1,16
             COUNT = COUNT + 1
             IPROD = I88PRO + I8PRO - 1
             LPROD(IPROD)=.TRUE.
             LFIRST=.TRUE.
50             ILL=IL
               CALL GETSYM(LBUF,IBUF,1,IC,IL,LFIX)
             IF( (ITYPE.EQ.1.OR.ITYPE.EQ.2.AND.IPRD.GT.13
  C                .AND.IPRD.LT.18).AND.COUNT.GT.2 ) LPRD=.TRUE.
             IF( (ITYPE.EQ.3.OR.ITYPE.EQ.2.AND.(IPRD.EQ.13.OR.
  C                IPRD.EQ.18)).AND.COUNT.GT.4 ) LPRD=.TRUE.
             IF( IOT.NE.A.AND.IOT.NE.C.AND.COUNT.GT.8 ) LPRD=.TRUE.
             IF( .NOT.LPRD ) GO TO 69
             IF(IL.NE.IFUNCT.AND.IL.NE.IDESC) ILL=IL
             IPROD = IPROD - 1
             GO TO 118
69             IF(LFIX) GO TO 59
               CALL MATCH(IMATCH,IBUF,ISYM)
               IF( ITYPE.EQ.1.AND.IMATCH.GT.11 ) LINP=.TRUE.
               IF( ITYPE.EQ.2.AND.(IMATCH.GT.12.AND.IMATCH.LT.19) )
  C                 LINP=.TRUE.
               IF( ITYPE.EQ.3.AND.(IMATCH.GT.13.AND.IMATCH.LT.18) )
  C                 LINP=.TRUE.
             ILL=IL
             IF(LINP) GO TO 100
             IF( IMATCH.EQ.0 ) GO TO 100
             IF( IMATCH.EQ.10.OR.IMATCH.EQ.99 ) GO TO 64
             IF(.NOT.LFIRST) GO TO 58
                 LFIRST=.FALSE.
                 DO 56 I=1,32
                     IBLOW = IBLOW + 1
56                   LFUSES(I,IPROD)=.TRUE.
58           CALL IXLATE(IINPUT,IMATCH,LPHASE,LBUF,ITYPE)
             IF(IINPUT.LE.0) GO TO 60
             IBLOW = IBLOW - 1
             LFUSES(IINPUT,IPROD)=.FALSE.
             CALL PLOT(LBUF,IBUF,LFUSES,IPROD,TITLE,.FALSE.,ITYPE,
  C                     LPROD,IOP,IBLOW)
             GO TO 60
59           CALL FIXSYM(LBUF,IBUF,IC,IL,LFIRST,LFUSES,IBLOW,
  C                       IPROD,LFIX)
60           IF(LAND) GO TO 50
64         IF(.NOT.LRIGHT) GO TO 68
66           CALL INCR(IC,IL,LFIX)
             IF(.NOT.LEQUAL)  GO TO 66
68         IF( .NOT.(LOR.OR.LEQUAL) ) GO TO 74
70         CONTINUE
74     ILL=IL
       CALL GETSYM(LBUF,IBUF,1,IC,IL,LFIX)
       IF(LLEFT.OR.LEQUAL) GO TO 28
100 IF( ILL.EQ.IFUNCT.OR.ILL.EQ.IDESC ) GO TO 102
    ILERR=ILL+4
    WRITE(1,101) (IBUF(I,1),I=1,8),ILERR,(IPAGE(I,ILL),I=1,79)
101 FORMAT(' ERROR SYMBOL = ',8A1,'     IN LINE NUMBER ',I3,
  C      /,' ',80A1)
    IF( (LACT).AND.(     LSAME).AND.(.NOT.LOPERR) )
    C       WRITE(1,103) IPAL,INOAI,IOT,INOO
 103 FORMAT(' OUTPUT MUST BE INVERTED SINCE ',4A1,A1,A1,A1,
    C      ' IS AN ACTIVE LOW DEVICE')
     IF( (LACT).AND.(.NOT.LSAME).AND.(.NOT.LOPERR) )
    C       WRITE(1,109) IPAL,INOAI,IOT,INOO
 109 FORMAT(' OUTPUT CANNOT BE INVERTED SINCE ',4A1,A1,A1,A1,
    C      ' IS AN ACTIVE HIGH DEVICE')
     IF( (LOPERR).AND.IMATCH.NE.0 )
    C       WRITE(1,105) IMATCH,IPAL,INOAI,IOT,INOO
 105 FORMAT(' THIS PIN NUMBER ',I2,' IS AN INVALID OUTPUT PIN',
    C      ' FOR ',4A1,A1,A1,A1)
     IF(LINP) WRITE(1,115) IMATCH,IPAL,INOAI,IOT,INOO
 115 FORMAT(' THIS PIN NUMBER ',I2,' IS AN INVALID INPUT PIN',
    C      ' FOR ',4A1,A1,A1,A1)
 118 ILERR=ILL+4
     IF(LPRD) WRITE(1,119)
    C (ISYM(I,IPRD),I=1,8),IPRD,ILERR,(IPAGE(I,ILL),I=1,79)
 119 FORMAT(' OUTPUT PIN NAME = ',8A1,' OUTPUT PIN NUMBER = ',I2,
    C      ' MINTERM IN LINE NUMBER ',I3,/,' ',80A1)
     IF( LPRD.AND.COUNT.LT.8 )
    C       WRITE(1,116) IPROD,IPAL,INOAI,IOT,INOO
 116 FORMAT(' THIS PRODUCT LINE NUMBER ',I2,' IS NOT VALID',
    C      ' FOR ',4A1,A1,A1,A1)
     IF( LPRD.AND.COUNT.GT.8 )
    C       WRITE(1,117) IPAL,INOAI,IOT,INOO
 117 FORMAT(' MAXIMUM OF 8 PRODUCTS LINES ARE VALID FOR ',4A1,A1,A1,A1,
    C      ' TOO MANY MINTERMS ARE SPECIFIED IN THIS EQUATION')
     STOP ERROR
 102 IF(ITYPE.LE.4) CALL TWEEK(ITYPE,IOT,LFUSES)
     ENDFILE 6
 108 WRITE(1,106)
 106 FORMAT(' OPERATION CODES:')
     WRITE(1,107)
 107 FORMAT(/,'  E=ECHO  O=PINOUT  P=PLOT    B=BRIEF    ',
    C       /,'  H=HEX   L=BHLF    N=BNPF    Q=QUIT    S=SIMULATE')
     WRITE(1,110)
 110 FORMAT(' ENTER OPERATION CODE:')
     READ(1,120) IOP
 120 FORMAT(A1)
     IF(IOP.EQ.E) CALL ECHO(IPAL,INOAI,IOT,INOO,REST,PATNUM,TITLE,
    C                       COMP)
     IF(IOP.EQ.O) CALL PINOUT(IPAL,INOAI,IOT,INOO,TITLE)
     IF(IOP.EQ.P) CALL PLOT(LBUF,IBUF,LFUSES,IPROD,TITLE,.TRUE.,ITYPE,
    C                       LPROD,IOP,IBLOW)
     IF(IOP.EQ.B) CALL PLOT(LBUF,IBUF,LFUSES,IPROD,TITLE,.TRUE.,ITYPE,
    C                       LPROD,IOP,IBLOW)
     IF(IOP.EQ.H) CALL HEX(LFUSES)
     IF(IOP.EQ.L) CALL BINR(LFUSES,H,L)
     IF(IOP.EQ.N) CALL BINR(LFUSES,P,N)
C    IF(IOP.EQ.R) GOTO 999
     IF(IOP.EQ.S) CALL TEST(LPHASE,LBUF,TITLE,IC,IL,ILE,ISYM,IBUF,
    C                       ITYPE,INOO,LFIX)
     IF(IOP.NE.Q) GO TO 108
     STOP
     END
C
C***********************************************************************
C
     SUBROUTINE INITLZ(INOAI,IOT,INOO,ITYPE,LFUSES,IC,IL,IBLOW,LFIX)
     BYTE    INOAI,IOT,INOO
     LOGICAL LBLANK,LLEFT,LAND,LOR,LSLASH,LEQUAL,LRIGHT,LXOR,LXNOR,
    C        LFIX,LFUSES(32,64)
       BYTE IPAGE,H,L,C,R,X,A,I0,I2,I4,I6,I8,INOAI,IOT,INOO
     COMMON  LBLANK,LLEFT,LAND,LOR,LSLASH,LEQUAL,LRIGHT,LXOR,LXNOR
     COMMON /PGE/ IPAGE(80,100)
     DATA H/'H'/,L/'L'/,C/'C'/,R/'R'/,X/'X'/,A/'A'/
    C     I0/'0'/,I2/'2'/,I4/'4'/,I6/'6'/,I8/'8'/
     DO 20 J=1,64
        DO 20 I=1,32
  20        LFUSES(I,J)=.FALSE.
     IBLOW=0
     IC=0
     IL=1
     ITYPE=0
     IF(  INOAI.EQ.I0 )                            ITYPE=1
```

continued

```
      IF(  INOAI.EQ.I2 )                             ITYPE=2
      IF(  INOAI.EQ.I4 )                             ITYPE=3
      IF( (INOAI.EQ.I6) )                            ITYPE=4
      IF( (INOAI.EQ.I6).AND.(INOO.EQ.I8) )           ITYPE=5
      IF( (IOT.EQ.R).OR.(IOT.EQ.X).OR.(IOT.EQ.A) )   ITYPE=6
      IF( .NOT.(IOT.EQ.H.OR.IOT.EQ.L.OR.IOT.EQ.C
      C       .OR.IOT.EQ.R.OR.IOT.EQ.X.OR.IOT.EQ.A) ) ITYPE=0
       CALL INCR(IC,IL,LFIX)
       RETURN
       END
C
C*************************************************************************
C
       SUBROUTINE INCR(IC,IL,LFIX)
       LOGICAL LBLANK,LLEFT,LAND,LOR,LSLASH,LEQUAL,LRIGHT,LXOR,LXNOR,
      C        LFIX,LX1
         BYTE  IPAGE,IBLANK,ILEFT,IAND,IOR,COMENT,ISLASH,IEQUAL,
      C        IRIGHT,ICOLON
       COMMON  LBLANK,LLEFT,LAND,LOR,LSLASH,LEQUAL,LRIGHT,LXOR,LXNOR
       COMMON /PGE/ IPAGE(80,100)
       DATA IBLANK/' '/,ILEFT/'('/,IAND/'*'/,IOR/'+'/,COMENT/';'/,
      C     ISLASH/'/'/,IEQUAL/'='/,IRIGHT/')'/,ICOLON/':'/
       LBLANK=.FALSE.
       LXOR=.FALSE.
       LXNOR=.FALSE.
       LX1=.FALSE.
       LRIGHT=.FALSE.
    10 IC=IC+1
       IF( IC.LE.79.AND.IPAGE(IC,IL).NE.COMENT ) GO TO 30
       IL=IL+1
    20 IC=0
       GO TO 10
    30 IF( IPAGE(IC,IL).EQ.ICOLON.AND.(LFIX) ) RETURN
       IF( IPAGE(IC,IL).NE.IBLANK ) GO TO 31
           LBLANK=.TRUE.
           GO TO 10
    31 IF( IPAGE(IC,IL).NE.ICOLON ) GO TO 32
       IF( (LXOR).OR.(LXNOR) )  GO TO 33
       LX1=.TRUE.
       GO TO 10
    33 IF(LXOR)  LOR=.TRUE.
       IF(LXNOR) LAND=.TRUE.
       RETURN
    32 IF( .NOT.(LX1.AND.(IPAGE(IC,IL).EQ.IOR.OR.IPAGE(IC,IL).EQ.IAND)) )
      C     GO TO 34
       IF( IPAGE(IC,IL).EQ.IOR  ) LXOR=.TRUE.
       IF( IPAGE(IC,IL).EQ.IAND ) LXNOR=.TRUE.
       GO TO 10
    34 LLEFT =( IPAGE(IC,IL).EQ.ILEFT  )
       LAND  =( IPAGE(IC,IL).EQ.IAND   )
       LOR   =( IPAGE(IC,IL).EQ. IOR   )
       LSLASH=( IPAGE(IC,IL).EQ.ISLASH )
       LEQUAL=( IPAGE(IC,IL).EQ.IEQUAL )
       LRIGHT=( IPAGE(IC,IL).EQ.IRIGHT )
       RETURN
       END
C
C*************************************************************************
C
       SUBROUTINE GETSYM(LPHASE,ISYM,J,IC,IL,LFIX)
       BYTE    ISYM(8,20)
       LOGICAL LBLANK,LLEFT,LAND,LOR,LSLASH,LEQUAL,LRIGHT,LXOR,LXNOR,
      C        LFIX,LPHASE(20)
         BYTE  IPAGE,IBLANK
       COMMON  LBLANK,LLEFT,LAND,LOR,LSLASH,LEQUAL,LRIGHT,LXOR,LXNOR
       COMMON /PGE/ IPAGE(80,100)
       DATA IBLANK/' '/
       LFIX=.FALSE.
       IF( .NOT.(LLEFT.OR.LAND.OR.LOR.OR.LEQUAL.OR.LRIGHT) ) GO TO 10
       CALL INCR(IC,IL,LFIX)
       IF(LLEFT) GO TO 60
    10 LPHASE(J)=( .NOT.LSLASH )
       IF(LPHASE(J)) GO TO 15
       CALL INCR(IC,IL,LFIX)
    15 DO 20 I=1,8
    20     ISYM(I,J)=IBLANK
```

```
   25 DO 30 I=1,7
   30     ISYM(I,J)=ISYM(I+1,J)
      ISYM(8,J)=IPAGE(IC,IL)
      CALL INCR(IC,IL,LFIX)
      IF( LLEFT.OR.LBLANK.OR.LAND.OR.LOR.OR.LRIGHT.OR.LEQUAL ) RETURN
      GO TO 25
   60 LFIX=.TRUE.
      RETURN
      END
C
C***************************************************************************
C
      SUBROUTINE MATCH(IMATCH,IBUF,ISYM)
      BYTE     IBUF(8,20),ISYM(8,20)
      LOGICAL LMATCH
        BYTE  C,A,R,Y
      DATA C/'C'/,A/'A'/,R/'R'/,Y/'Y'/
      IMATCH=0
      DO 20 J=1,20
          LMATCH=.TRUE.
          DO 10 I=1,8
   10         LMATCH=LMATCH.AND.(IBUF(I,1).EQ.ISYM(I,J))
          IF(LMATCH) IMATCH=J
   20     CONTINUE
      IF( IBUF(3,1).EQ.C.AND.IBUF(4,1).EQ.A.AND.IBUF(5,1).EQ.R.AND.
     C    IBUF(6,1).EQ.R.AND.IBUF(7,1).EQ.Y ) IMATCH=99
      RETURN
      END
C
C**********************************************************************
C
      SUBROUTINE IXLATE(IINPUT,IMATCH,LPHASE,LBUF,ITYPE)
      BYTE     ITABLE(20,6)
      LOGICAL LPHASE(20),LBUF(20)
      DATA     ITABLE/
     C   3, 1, 5, 9,13,17,21,25,29,-10,31,-1,-1,-1,-1,-1,-1,-1,-1,-20,
     C   3, 1, 5, 9,13,17,21,25,29,-10,31,27,-1,-1,-1,-1,-1,-1, 7,-20,
     C   3, 1, 5, 9,13,17,21,25,29,-10,31,27,23,-1,-1,-1,-1,11, 7,-20,
     C   3, 1, 5, 9,13,17,21,25,29,-10,31,27,23,19,-1,-1,15,11, 7,-20,
     C   3, 1, 5, 9,13,17,21,25,29,-10,31,-1,27,23,19,15,11, 7,-1,-20,
     C  -1, 1, 5, 9,13,17,21,25,29,-10,-1,31,27,23,19,15,11, 7, 3,-20/
      IINPUT=0
      IBUBL=0
      IF(((     LPHASE(IMATCH)).AND.(.NOT.LBUF(1))).OR.
     C   ((.NOT.LPHASE(IMATCH)).AND.(     LBUF(1)))) IBUBL=1
      IF( ITABLE(IMATCH,ITYPE).GT.0 ) IINPUT=ITABLE(IMATCH,ITYPE)+IBUBL
      RETURN
      END
C
C************************************************************************
C
      SUBROUTINE PLOT(LBUF,IBUF,LFUSES,IPROD,TITLE,LDUMP,ITYPE,
     C                LPROD,IOP,IBLOW)
      BYTE     IBUF(8,20),IOUT(64),TITLE(80)
      LOGICAL LBUF(20),LFUSES(32,64),LDUMP,LPROD(80)
        BYTE  ISAVE(64,32),IAND,IOR,ISLASH,
     C        IDASH,X,IBLANK,P,B,HIFANT,IOP,CLRS
      DATA ISAVE/2048*' '/,IAND/'*'/,IOR/'+'/,ISLASH/'/'/,
     C     IDASH/'-'/,X/'X'/,IBLANK/' '/,P/'P'/,B/'B'/,
     C     HIFANT/'O'/,CLRS/X'0C'/
      IF(.NOT.LDUMP) GO TO 4
    4 IF(LDUMP) GO TO 60
      IF(ISAVE(IPROD,1).NE.IBLANK) RETURN
      IF( LBUF(1) ) GO TO 5
      DO 30 J=1,31
   30     ISAVE(IPROD,J)=ISAVE(IPROD,J+1)
      ISAVE(IPROD,32)=ISLASH
    5 DO 20 I=1,8
          IF( ISAVE(IPROD,1).NE.IBLANK ) RETURN
           IF( IBUF(I,1).EQ.IBLANK ) GO TO 20
           DO 10 J=1,31
   10          ISAVE(IPROD,J)=ISAVE(IPROD,J+1)
           ISAVE(IPROD,32)=IBUF(I,1)
   20      CONTINUE
      IF(ISAVE(IPROD,1).NE.IBLANK) RETURN
```

continued

```
  40 DO 50 J=1,31
  50     ISAVE(IPROD,J)=ISAVE(IPROD,J+1)
     ISAVE(IPROD,32)=IAND
     RETURN
  60 WRITE(1,62) CLRS,TITLE
  62 FORMAT(' ',A1,80A1,//,
    C '                 11 1111 1111 2222 2222 2233',/,
    C '     0123 4567 8901 2345 6789 0123 4567 8901',/)
     DO 100 I88PRO=1,57,8
         DO 94 I8PRO=1,8
             IPROD=I88PRO+I8PRO-1
             ISAVE(IPROD,32)=IBLANK
             DO 70 I=1,32
                 IF( ISAVE(IPROD,1).NE.IBLANK ) GO TO 70
                 DO 65 J=1,31
                     ISAVE(IPROD,J)=ISAVE(IPROD,J+1)
  65                 CONTINUE
                  ISAVE(IPROD,32)=IBLANK
   70         CONTINUE
              DO 80 I=1,32
                  IOUT(I)=X
                  IF( LFUSES(I,IPROD) ) IOUT(I)=IDASH
                  IOUT(I+32)=ISAVE(IPROD,I)
   80         CONTINUE
              IF(ITYPE.LE.4) CALL FANTOM(ITYPE,IOUT,IPROD,I8PRO)
              IPROD=IPROD-1
              DO 85 J=1,32
                  IF( IOP.EQ.B.AND.IOUT(J).EQ.HIFANT ) IOUT(J)=IBLANK
   85         CONTINUE
              IF( (IOP.EQ.P).OR.(IOP.EQ.B.AND.(LPROD(IPROD+1))) )
     C        WRITE(1,90) IPROD,IOUT
   90         FORMAT(' ',I2,8(' ',4A1),' ',32A1)
   94         CONTINUE
          WRITE(1,96)
   96     FORMAT(1X)
  100     CONTINUE
      WRITE(1,110)
  110 FORMAT(/,
     C' LEGEND:  X : FUSE NOT BLOWN (L,N,0)   - : FUSE BLOWN   (H,P,1)')
      IF( IOP.EQ.P.AND.ITYPE.LE.4 ) WRITE(1,111)
  111 FORMAT(
     C'          0 : PHANTOM FUSE   (L,N,0)   O : PHANTOM FUSE (H,P,1)')
      WRITE(1,112) IBLOW
  112 FORMAT(/,' NUMBER OF FUSES BLOWN = ',I4)
      WRITE(1,113)
  113 FORMAT(////)
      RETURN
      END
C
C***********************************************************************
C
      SUBROUTINE TWEEK(ITYPE,IOT,LFUSES)
      BYTE    IOT
      LOGICAL LFUSES(32,64)
        BYTE  L,C
      DATA L/'L'/,C/'C'/
      IF(ITYPE.GE.4) GO TO 20
      DO 10 IPROD=1,64
          LFUSES(15,IPROD)=.TRUE.
          LFUSES(16,IPROD)=.TRUE.
          LFUSES(19,IPROD)=.TRUE.
          LFUSES(20,IPROD)=.TRUE.
          IF(ITYPE.GE.3) GO TO 10
          LFUSES(11,IPROD)=.TRUE.
          LFUSES(12,IPROD)=.TRUE.
          LFUSES(23,IPROD)=.TRUE.
          LFUSES(24,IPROD)=.TRUE.
          IF(ITYPE.GE.2) GO TO 10
          LFUSES( 7,IPROD)=.TRUE.
          LFUSES( 8,IPROD)=.TRUE.
          LFUSES(27,IPROD)=.TRUE.
          LFUSES(28,IPROD)=.TRUE.
   10     CONTINUE
      DO 18 IINPUT=7,28
          DO 12 IPROD=1,57,8
              LFUSES(IINPUT,IPROD+4)=.FALSE.
              LFUSES(IINPUT,IPROD+5)=.FALSE.
              LFUSES(IINPUT,IPROD+6)=.FALSE.
   12         LFUSES(IINPUT,IPROD+7)=.FALSE.
          IF(ITYPE.GE.3) GO TO 18
          DO 14 IPROD=17,41,8
              LFUSES(IINPUT,IPROD+2)=.FALSE.
   14         LFUSES(IINPUT,IPROD+3)=.FALSE.
          IF(ITYPE.GE.2) GO TO 18
          DO 16 IPROD=1,57,8
              LFUSES(IINPUT,IPROD+2)=.FALSE.
   16         LFUSES(IINPUT,IPROD+3)=.FALSE.
   18 CONTINUE
   20 IF( (ITYPE.EQ.1) .OR. ((ITYPE.EQ.4).AND.(IOT.EQ.L)) ) RETURN
      DO 99 IINPUT=1,32
          DO 30 IPROD=1,8
              LFUSES(IINPUT,IPROD+ 0)= (IOT.NE.L)
   30         IF(IOT.NE.C) LFUSES(IINPUT,IPROD+56)= (IOT.NE.L)
          IF(ITYPE.LE.2) GO TO 99
          DO 40 IPROD=1,8
              LFUSES(IINPUT,IPROD+ 8)= (IOT.NE.L)
   40         IF(IOT.NE.C) LFUSES(IINPUT,IPROD+48)= (IOT.NE.L)
          IF(ITYPE.LE.3) GO TO 99
          DO 50 IPROD=1,8
              LFUSES(IINPUT,IPROD+16)= (IOT.NE.L)
   50         IF(IOT.NE.C) LFUSES(IINPUT,IPROD+40)= (IOT.NE.L)
   99     CONTINUE
      RETURN
      END
C
C************************************************************************
C
      SUBROUTINE SLIP(LFUSES,I88PRO,INOAI,IOT,INOO,IBLOW)
      LOGICAL LFUSES(32,64)
        BYTE  R,I1,I2,I4,I6,I8,IOT,INOO,INOAI
      DATA R/'R'/,I1/'1'/,I2/'2'/,I4/'4'/,I6/'6'/,I8/'8'/
      IF( (INOAI.NE.I6) .OR. (INOO.EQ.I1) .OR.  (INOO.EQ.I2) .OR.
     C    ( (IOT.EQ.R).AND.(INOO.EQ.I8) ) .OR.
     C    ( (I88PRO.GE. 9).AND.(I88PRO.LE.49).AND.(INOO.EQ.I6) ) .OR.
     C    ( (I88PRO.GE.17).AND.(I88PRO.LE.41).AND.(INOO.EQ.I4)) ) RETURN
      DO 10 I=1,32
      IBLOW = IBLOW + 1
   10 LFUSES(I,I88PRO) = .TRUE.
      I88PRO = I88PRO + 1
      RETURN
      END
C
C************************************************************************
C
      SUBROUTINE FANTOM(ITYPE,IOUT,IPROD,I8PRO)
      BYTE    IOUT(64)
        BYTE  X,IDASH,LOFANT,HIFANT
      DATA X/'X'/,IDASH/'-'/,LOFANT/'0'/,HIFANT/'O'/
      DO 10 I=1,32
          IF( IOUT(I).EQ.IDASH ) IOUT(I)=HIFANT
          IF( IOUT(I).EQ.X )     IOUT(I)=LOFANT
   10 CONTINUE
       IF((ITYPE.EQ.4).AND.((IPROD.LE.24).OR.(IPROD.GE.41))) RETURN
       IF((ITYPE.EQ.3).AND.((IPROD.LE.16).OR.(IPROD.GE.45))) RETURN
       IF((ITYPE.EQ.2).AND.((IPROD.LE. 8).OR.(IPROD.GE.53))) RETURN
       IF((ITYPE.LE.3).AND.(I8PRO.GE.5)) RETURN
       IF((ITYPE.LE.2).AND.(IPROD.GE.19).AND.(IPROD.LE.48).AND.
     C    (I8PRO.GE.3)) RETURN
       IF((ITYPE.EQ.1).AND.(I8PRO.GE.3)) RETURN
       DO 50 I=1,32
         IF(((I.EQ.15).OR.(I.EQ.16).OR.(I.EQ.19).OR.(I.EQ.20)).AND.
     C    (ITYPE.LE.3)) GO TO 50
         IF(((I.EQ.11).OR.(I.EQ.12).OR.(I.EQ.23).OR.(I.EQ.24)).AND.
     C    (ITYPE.LE.2)) GO TO 50
         IF(((I.EQ. 7).OR.(I.EQ. 8).OR.(I.EQ.27).OR.(I.EQ.28)).AND.
     C    (ITYPE.LE.1)) GO TO 50
         IF( IOUT(I).EQ.HIFANT ) IOUT(I)=IDASH
         IF( IOUT(I).EQ.LOFANT ) IOUT(I)=X
    50 CONTINUE
       RETURN
       END
```

continued

```
C
C****************************************************************************
C       *****************************************************************
        SUBROUTINE DATAIO (TEXT,NUMBER)
        LOGICAL TEXT(1)
        INTEGER NUMBER
        EXTERNAL PUNCH
        DO 10 I= 1, NUMBER
 10     CALL PUNCH(TEXT(I))
        RETURN
        END
C      ***********************************************************
C      ***********************************************************
C      ***********************************************************
       LOGICAL FUNCTION IHEXA(I)
       LOGICAL STRNG(16)
       DATA STRNG/'0','1','2','3','4','5','6','7','8','9',
     1  'A','B','C','D','E','F'/
       M=MOD(I,16)+1
       IHEXA=STRNG(M)
       RETURN
       END
C    **********
       SUBROUTINE HEX(LFUSES)
       LOGICAL LFUSES(32,64)
       LOGICAL    ITEMP(64),IHEXA
        LOGICAL T(128)
        LOGICAL STX,ETX,NULL(50),DC1,READER
        EXTERNAL READER
        DATA STX/X'02'/,ETX/X'03'/,NULL/50*X'00'/,DC1/X'11'/
        WRITE(1,81)
  81    FORMAT(' DATA I/O SETUP:'/' TYPE ''SELECT 50,ENTER''')
        WRITE(1,82)
  82    FORMAT(' TYPE ''SELECT D2,ENTER''')
        WRITE(1,83)
  83    FORMAT(' THEN PRESS ''START'' BUTTON ')
  87    IF(READER(0).XOR.DC1) GOTO 87
        WRITE(1,88)
  88    FORMAT(' STARTING TRANSMISSION')
        ENCODE(T,70)STX
        CALL DATAIO(T,1)
       DO 40 I=1,33,32
       INC=I-1
       DO 40 IPROD=1,7,2
       DO 20 J=1,2
       DO 20 IINPUT=1,32
       IHEX=0
       M=IPROD+INC+J-1
         IF(LFUSES(IINPUT,M+ 0)) IHEX=IHEX+1
         IF(LFUSES(IINPUT,M+ 8)) IHEX=IHEX+2
         IF(LFUSES(IINPUT,M+16)) IHEX=IHEX+4
         IF(LFUSES(IINPUT,M+24)) IHEX=IHEX+8
       M=IINPUT+32*(J-1)
  20   ITEMP(M)=IHEXA(IHEX)
        ENCODE(T,60)ITEMP
  40    CALL DATAIO(T,128)
        ENCODE(T,80)ETX,NULL
        CALL DATAIO(T,51)
  60   FORMAT(64(A1,' '))
  70    FORMAT(A1)
  80    FORMAT(51A1)
       RETURN
       END
C
C*************************************************************************
C
      SUBROUTINE ECHO(IPAL,INOAI,IOT,INOO,REST,PATNUM,TITLE,COMP)
      BYTE    IPAL(4),REST(73),PATNUM(80),TITLE(80),COMP(79)
        BYTE  IPAGE,INOAI,IOT,INOO,CLRS
      COMMON /PGE/ IPAGE(80,100)
      COMMON /FTEST/ IFUNCT,IDESC,IEND
        DATA  CLRS/X'0C'/
      WRITE(1,10)CLRS,IPAL,INOAI,IOT,INOO,REST,PATNUM,TITLE,COMP
   10 FORMAT(' ',A1,4A1,A1,A1,A1,73A1,/,' ',80A1,/,' ',80A1,/,' ',80A1)
      DO 30 J=1,IEND
           WRITE(1,20) (IPAGE(I,J),I=1,80)
   20      FORMAT(' ',80A1)
   30 CONTINUE
      RETURN
      END
C
C*****************************************************************************
*
C
      SUBROUTINE BINR(LFUSES,H,L)
      BYTE    ITEMP(4,8),H,L,CLRS
      LOGICAL LFUSES(32,64)
        DATA CLRS/X'0C'/
      WRITE(1,10)CLRS
   10 FORMAT(' ',A1)
      DO 20 I=1,33,32
      INC=I-1
        DO 20 IPROD=1,8
            DO 20 J=1,25,8
              DO 15 K=1,8
                IINPUT=J+K-1
                ITEMP(1,K)=L
                ITEMP(2,K)=L
                ITEMP(3,K)=L
                ITEMP(4,K)=L
                MYINX = IPROD + INC
                IF(LFUSES(IINPUT,MYINX +  0)) ITEMP(4,K)=H
                IF(LFUSES(IINPUT,MYINX +  8)) ITEMP(3,K)=H
                IF(LFUSES(IINPUT,MYINX + 16)) ITEMP(2,K)=H
                IF(LFUSES(IINPUT,MYINX + 24)) ITEMP(1,K)=H
   15           CONTINUE
   20         WRITE(1,30) ITEMP
   30         FORMAT(' ',8('B',4A1,'F '))
      WRITE(1,10)
      RETURN
      END
C
C***************************************************************************
C
      SUBROUTINE PINOUT(IPAL,INOAI,IOT,INOO,TITLE)
      BYTE    IPAL(4),TITLE(80),PIN(8,20),IIN(7,2)
        BYTE  IPAGE,IBLANK,ISTAR,INOAI,IOT,INOO,CLRS
      COMMON /PGE/ IPAGE(80,100)
      DATA IBLANK/' '/,ISTAR/'*'/,CLRS/X'0C'/
      DO 10 J=1,20
          DO 5 I=1,8
    5         PIN(I,J)=IBLANK
   10 CONTINUE
   15 DO 25 J=1,2
          DO 20 I=1,7
   20         IIN(I,J)=IBLANK
   25 CONTINUE
      IIN(2,1)=IPAL(1)
      IIN(4,1)=IPAL(2)
      IIN(6,1)=IPAL(3)
      IIN(1,2)=IPAL(4)
      IIN(3,2)=INOAI
      IIN(5,2)=IOT
      IIN(7,2)=INOO
      J=0
      IL=0
   30 IC=0
      IL=IL+1
   35 IC=IC+1
   40 IF( IC.GT.80 ) GO TO 30
      IF( IPAGE(IC,IL).EQ.IBLANK ) GO TO 35
      J=J+1
      IF(J.GT.20) GO TO 60
      DO 55 I=1,8
          PIN(I,J)=IPAGE(IC,IL)
          IC=IC+1
          IF( IC.GT.80 ) GO TO 40
          IF( IPAGE(IC,IL).EQ.IBLANK ) GO TO 40
   55 CONTINUE
   60 DO 75 J=1,10
          II=0
```

continued

```
 65      II=II+1
         IF(II.EQ.9) GO TO 75
         IF( PIN(II,J).NE.IBLANK ) GO TO 65
         I=9
 70      I=I-1
         II=II-1
         PIN(I,J)=PIN(II,J)
         PIN(II,J)=IBLANK
         IF(II.NE.1) GO TO 70
 75 CONTINUE
    WRITE(1,76)CLRS,TITLE
 76 FORMAT(' ',A1,80A1)
    WRITE(1,78) ISTAR,ISTAR,ISTAR,ISTAR,ISTAR,ISTAR,ISTAR,ISTAR,
   C            ISTAR,ISTAR,ISTAR,ISTAR,ISTAR,ISTAR,ISTAR,ISTAR,
   C            ISTAR,ISTAR,ISTAR,ISTAR,ISTAR,ISTAR,ISTAR,ISTAR,
   C            ISTAR,ISTAR,ISTAR,ISTAR,ISTAR,ISTAR,ISTAR,ISTAR
 78 FORMAT(/,' ',14X,14A1,3X,14A1,
   C       /,' ',14X,A1,13X,A1,1X,A1,13X,A1)
    JJ=20
    DO 88 J=1,10
        WRITE(1,80) ISTAR,ISTAR,ISTAR,ISTAR,ISTAR,ISTAR,ISTAR,ISTAR
 80 FORMAT(' ',11X,4A1,29X,4A1)
    WRITE(1,81) (PIN(I,J),I=1,8),ISTAR,J,ISTAR,
   C     (IIN(I,1),I=1,7),ISTAR,JJ,ISTAR,(PIN(I,JJ),I=1,8)
 81 FORMAT(' ',8A1,3X,A1,I2,A1,11X,7A1,11X,A1,I2,A1,3X,8A1)
    WRITE(1,82) ISTAR,ISTAR,ISTAR,ISTAR,ISTAR,ISTAR,ISTAR,ISTAR
 82 FORMAT(' ',11X,4A1,29X,4A1)
    WRITE(1,84) ISTAR,(IIN(I,2),I=1,7),ISTAR
 84 FORMAT(' ',14X,A1,11X,7A1,11X,A1)
    DO 86 II=1,2
        DO 85 I=1,7
 85         IIN(I,II)=IBLANK
 86 CONTINUE
    JJ=JJ-1
 88 CONTINUE
    WRITE(1,90) ISTAR,ISTAR,ISTAR,ISTAR,ISTAR,ISTAR,ISTAR,ISTAR,
   C            ISTAR,ISTAR,ISTAR,ISTAR,ISTAR,ISTAR,ISTAR,ISTAR,
   C            ISTAR,ISTAR,ISTAR,ISTAR,ISTAR,ISTAR,ISTAR,ISTAR,
   C            ISTAR,ISTAR,ISTAR,ISTAR,ISTAR,ISTAR,ISTAR
 90 FORMAT(' ',14X,31A1)
    RETURN
    END
C
C***************************************************************************
C
      SUBROUTINE FIXSYM(LBUF,IBUF,IC,IL,LFIRST,LFUSES,IBLOW,IPROD,LFIX)
      LOGICAL LBUF(20),LFUSES(32,64),LFIRST,LMATCH,LFIX
      BYTE    IBUF(8,20),FIXBUF(8)
        BYTE  IPAGE,A,B,ISLASH,IOR,IBLANK,IRIGHT,IAND,
     C        N,Q,N0,N1,N2,N3,ICOLON,TABLE(5,14)
      COMMON /PGE/ IPAGE(80,100)
      DATA A/'A'/,B/'B'/,ISLASH/'/'/,IOR/'+'/,IBLANK/' '/,IRIGHT/')'/,
     C        IAND/'*'/,N/'N'/,Q/'Q'/,N0/'0'/,N1/'1'/,N2/'2'/,N3/'3'/,
     C      ICOLON/':'/
      DATA    TABLE      /      ' ','A','+','/','B',' ',' ','A','+','B',
     C      ' ',' ',' ',' ','A','/','A','+','/','B',' ',' ',' ','/','B',
     C      'A',':','+',':','B',' ','A','*','/','B',' ','/','A','+','B',
     C      'A',':','*',':','B',' ',' ',' ',' ','B',' ',' ','A','*','B',
     C      ' ',' ',' ','/','A','/','A','*','/','B',' ','/','A','*','B'/
      IINPUT=0
      DO 20 I=1,8
          IBUF(I,1)=IBLANK
  20      FIXBUF(I)=IBLANK
  21  CALL INCR(IC,IL,LFIX)
      I=IPAGE(IC,IL)
      IF(I.EQ.IRIGHT) GO TO 40
      IF(I.EQ.N0) IINPUT=8
      IF(I.EQ.N1) IINPUT=12
      IF(I.EQ.N2) IINPUT=16
      IF(I.EQ.N3) IINPUT=20
      DO 24 J=1,7
  24      IBUF(J,1)=IBUF(J+1,1)
      IBUF(8,1)=I
      IF(.NOT. ( (I.EQ.A).OR.(I.EQ.B).OR.(I.EQ.ISLASH).OR.(I.EQ.IOR)
     C      .OR.(I.EQ.IAND).OR.(I.EQ.ICOLON) ) )  GO TO 21
      DO 30 I=1,4
```

```
30      FIXBUF(I)=FIXBUF(I+1)
   FIXBUF(5)=IPAGE(IC,IL)
   GO TO 21
40 IMATCH=0
   DO 60 J=1,14
       LMATCH=.TRUE.
       DO 50 I=1,5
50          LMATCH=LMATCH .AND. ( FIXBUF(I).EQ.TABLE(I,J) )
60      IF(LMATCH) IMATCH=J
   IF(IMATCH.EQ.0) GO TO 100
   IF(.NOT.LFIRST) GO TO 85
       LFIRST=.FALSE.
       DO 80 I=1,32
           LFUSES(I,IPROD)=.TRUE.
80         IBLOW = IBLOW + 1
85 DO 90 I=1,4
       IF( (IMATCH-7).LE.0 ) GO TO 90
       MYINX = IINPUT + I
       LFUSES(MYINX,IPROD)=.FALSE.
       IBLOW = IBLOW - 1
       IMATCH=IMATCH-8
90 IMATCH=IMATCH+IMATCH
   LBUF(1)=.TRUE.
   CALL PLOT(LBUF,IBUF,LFUSES,IPROD,TITLE,.FALSE.,ITYPE,
  C         LPROD,IOP,IBLOW)
100 LFIX=.FALSE.
   CALL INCR(IC,IL,LFIX)
   RETURN
   END
```

SIMLT.FOR
Contributed by: Trevor G. Marshall
"PALs Simplify Complex Circuits," by Trevor G. Marshall. January, page 247.

```
C
C**********************************************************************
C
      SUBROUTINE TEST(LPHASE,LBUF,TITLE,IC,IL,ILE,ISYM,IBUF,
     C                ITYPE,INOO,LFIX)
      BYTE    ISYM(8,20),ISYM1(8,20),IBUF(8,20),
     C        IVECT(20),IVECTP(20),IPAGE,IDASH,L,H,X,C,Z,N0,N1,
     C        IBLANK,COMENT,I6,I8,CLRS,INOO,XORSUM,
     C        ISTATE(20),ISTATT(20),IPIN(20),TITLE(80)
      LOGICAL LBLANK,LLEFT,LAND,LOR,LSLASH,LEQUAL,LRIGHT,LXOR,LXNOR,
     C        LFIX,LSAME,XORFND,LERR,LPHASE(20),LPHAS1(20),LBUF(20),
     C        LOUT(20),LOUTP(20),LCLOCK,LPTRST,LCTRST,LENABL(20),NREG
      COMMON  LBLANK,LLEFT,LAND,LOR,LSLASH,LEQUAL,LRIGHT,LXOR,LXNOR
      COMMON /PGE/ IPAGE(80,100)
      COMMON /FTEST/ IFUNCT,IDESC,IEND
      DATA IDASH/'-'/,L/'L'/,H/'H'/,X/'X'/,C/'C'/,Z/'Z'/,
     C    N0/'0'/,N1/'1'/,
     C    IBLANK/' '/,COMENT/';'/,I6/'6'/I8/'8'/,CLRS/X'0C'/
      IF(IFUNCT.NE.0) GO TO 3
      WRITE(1,2)
    2 FORMAT(/,' FUNCTION TABLE MUST BE SUPPLIED IN ORDER TO PERFORM',
     C        ' SIMULATION')
      RETURN
    3 WRITE(1,4)CLRS,TITLE
    4 FORMAT(' ',A1,' CHECKING THE FUNCTION TABLE',80A1,/)
      LERR=.FALSE.
      ITRST=0
      IC=0
      IL=IFUNCT + 1
      CALL INCR(IC,IL,LFIX)
      DO 10 I=1,19
      CALL GETSYM(LPHAS1,ISYM1,I,IC,IL,LFIX)
         DO 5 J=1,8
    5    IBUF(J,1)=ISYM1(J,I)
      IF(IBUF(8,1).EQ.IDASH) GO TO 12
      CALL MATCH(IMATCH,IBUF,ISYM)
      IF(IMATCH.NE.0) GO TO 7
      WRITE(1,6) (IBUF(J,1),J=1,8)
```

continued

```
 6 FORMAT(/,' FUNCTION TABLE PIN LIST ERROR AT', 8A1)
   RETURN
 7 LOUT(I)=.FALSE.
   ISTATT(I)=X
   IVECTP(I)=X
   IF(ITYPE.NE.6) GO TO 10
   IF(IMATCH.EQ.1)  ICLOCK=I
   IF(IMATCH.EQ.11) ITRST=I
10 IPIN(I)=IMATCH
12 IMAX=I-1
   NVECT=0
90 NVECT=NVECT+1
   IC1=0
   IL1=ILE
23 IF(IPAGE(1,IL).NE.COMENT) GO TO 24
   IL=IL+1
   GO TO 23
24 CONTINUE
   DO 20 I=1,IMAX
     IF(IPAGE(IC,IL).EQ.IBLANK) GO TO 21
     GO TO 22
21   IC=IC+1
     IF(IPAGE(IC,IL).EQ.IBLANK) GO TO 21
22   IVECT(I)=IPAGE(IC,IL)
     IC=IC+1
20 CONTINUE
   IL=IL+1
   IC=1
   IF(IVECT(1).EQ.IDASH) GO TO 95
   DO 11 I=1,IMAX
      IF( IVECT(I).EQ.L.OR.IVECT(I).EQ.H.OR.IVECT(I).EQ.X.OR.
  C         IVECT(I).EQ.Z.OR.IVECT(I).EQ.C) GO TO 11
      WRITE(1,8) IVECT(I),NVECT
 8    FORMAT(/,' ',A1,' IS NOT AN ALLOWED FUNCTION TABLE ENTRY',
  C                      ' IN VECTOR ',I3)
      RETURN
11 CONTINUE
   LCLOCK=.FALSE.
   LCTRST=.TRUE.
   LPTRST=.TRUE.
   DO 13 I=1,IMAX
13    LENABL(I)=.TRUE.
   NREG=.FALSE.
   DO 15 I=1,20
15 ISTATE(I)=X
   IF(ITYPE.NE.6) GO TO 25
   IF(IVECT(ICLOCK).EQ.C) LCLOCK=.TRUE.
   LSAME=( (      LPHASE(11)).AND.(      LPHAS1(ITRST)).OR.
  C        (.NOT.LPHASE(11)).AND.(.NOT.LPHAS1(ITRST)) )
   IF( IVECT(ITRST).EQ.L.AND.(.NOT.LSAME).OR.
  C    IVECT(ITRST).EQ.H.AND.(      LSAME) ) LPTRST=.FALSE.
   IF(LPTRST) GO TO 25
   DO 46 I=1,IMAX
   J=IPIN(I)
      IF(J.EQ.14.OR.J.EQ.15.OR.J.EQ.16.OR.J.EQ.17) LENABL(I)=.FALSE.
      IF( INOO.EQ.I6.AND.(J.EQ.13.OR.J.EQ.18) )    LENABL(I)=.FALSE.
      IF( INOO.EQ.I8.AND.(J.EQ.12.OR.J.EQ.13
  C                      .OR.J.EQ.18.OR.J.EQ.19) )    LENABL(I)=.FALSE.
46 CONTINUE
25 CALL INCR(IC1,IL1,LFIX)
26 CALL GETSYM(LBUF,IBUF,1,IC1,IL1,LFIX)
   IF(LLEFT) GO TO 29
27 IF(.NOT.LEQUAL) GO TO 26
29 IF(LEQUAL) GO TO 35
   NREG=.TRUE.
33 CALL GETSYM(LBUF,IBUF,1,IC1,IL1,LFIX)
   CALL MATCH(IINP,IBUF,ISYM1)
   IF(IINP.NE.0) GO TO 32
   CALL MATCH(IMATCH,IBUF,ISYM)
   ILL=IL1
   IF( IINP.EQ.0.AND.IMATCH.NE.10.AND.IMATCH.NE.20 ) GO TO 100
   IF( IMATCH.EQ.10.AND.(LBUF(1)).OR.
  C    IMATCH.EQ.20.AND.(.NOT.LBUF(1)) ) LCTRST=.FALSE.
   GO TO 34
32 ITEST=IVECT(IINP)
   IF(  ITEST.EQ.L.AND.(      LPHAS1(IINP)).AND.(      LBUF(1))
  C.OR. ITEST.EQ.H.AND.(     LPHAS1(IINP)).AND.(.NOT.LBUF(1))
  C.OR. ITEST.EQ.H.AND.(.NOT.LPHAS1(IINP)).AND.(     LBUF(1))
  C.OR. ITEST.EQ.L.AND.(.NOT.LPHAS1(IINP)).AND.(.NOT.LBUF(1))
  C  )  LCTRST=.FALSE.
   IF(ITEST.EQ.X.OR.ITEST.EQ.Z) LCTRST=.FALSE.
34 IF(LAND) GO TO 33
   GO TO 27
35 CALL MATCH(IOUTP,IBUF,ISYM1)
   ILL=IL1
   IF(IOUTP.EQ.0) GO TO 100
   IF(NREG) LENABL(IOUTP)=LCTRST
   LOUT(IOUTP)=.TRUE.
   IF( .NOT.LCTRST ) LOUT(IOUTP)=.FALSE.
   LCTRST=.TRUE.
   LOUTP(IOUTP)=LBUF(1)
   XORSUM=H
   XORFND=.FALSE.
   ISUM=L
28 IPROD=H
30 ILL=IL1
   CALL GETSYM(LBUF,IBUF,1,IC1,IL1,LFIX)
   IF( .NOT.LFIX ) GO TO 39
       LFIX=.FALSE.
       CALL FIXTST(LPHAS1,LBUF,IC1,IL1,ISYM,ISYM1,IBUF,
  C                IVECT,IVECTP,ITEST,LCLOCK,NREG,LFIX)
       IF(IPROD.EQ.H) IPROD=ITEST
       GO TO 38
39 CALL MATCH(IINP,IBUF,ISYM1)
   IF(IINP.NE.0) GO TO 45
   CALL MATCH(IMATCH,IBUF,ISYM)
   IF(IMATCH.NE.10) GO TO 100
   ITEST=L
   IINP=19
   LPHAS1(19)=.TRUE.
   GO TO 37
45 ITEST=IVECT(IINP)
   IF( (.NOT.LCLOCK).OR.(NREG) ) GO TO 37
   CALL MATCH(IIFB,IBUF,ISYM)
   IF( IIFB.EQ.14.OR.IIFB.EQ.15.OR.IIFB.EQ.16.OR.IIFB.EQ.17 )
  C     ITEST=IVECTP(IINP)
   IF( (INOO.EQ.I6.OR.INOO.EQ.I8).AND.(IIFB.EQ.13.OR.IIFB.EQ.18) )
  C     ITEST=IVECTP(IINP)
   IF( INOO.EQ.I8.AND.(IIFB.EQ.12.OR.IIFB.EQ.19) )
  C     ITEST=IVECTP(IINP)
37 IF(  ITEST.EQ.L.AND.(     LPHAS1(IINP)).AND.(     LBUF(1))
  C.OR. ITEST.EQ.H.AND.(     LPHAS1(IINP)).AND.(.NOT.LBUF(1))
  C.OR. ITEST.EQ.H.AND.(.NOT.LPHAS1(IINP)).AND.(     LBUF(1))
  C.OR. ITEST.EQ.L.AND.(.NOT.LPHAS1(IINP)).AND.(.NOT.LBUF(1))
  C  )  IPROD=L
38 IF(LRIGHT) CALL INCR(IC1,IL1,LFIX)
   IF(LAND) GO TO 30
   IF(ISUM.EQ.L.AND.IPROD.EQ.X) ISUM=X
   IF( (ISUM.NE.H).AND.IPROD.EQ.H ) ISUM=H
   IF(.NOT.LXOR) GO TO 31
   XORSUM=ISUM
   XORFND=.TRUE.
   ISUM=L
   GO TO 28
31 IF(LOR) GO TO 28
   IF(.NOT.XORFND)    ISTATT(IOUTP)=ISUM
   IF( (XORFND).AND.((ISUM.EQ.L.AND.XORSUM.EQ.L).OR.
  C                  (ISUM.EQ.H.AND.XORSUM.EQ.H)) ) ISTATT(IOUTP)=L
   IF( (XORFND).AND.((ISUM.EQ.H.AND.XORSUM.EQ.L).OR.
  C                  (ISUM.EQ.L.AND.XORSUM.EQ.H)) ) ISTATT(IOUTP)=H
   IF( (XORFND).AND. (ISUM.EQ.X.OR. XORSUM.EQ.X) )  ISTATT(IOUTP)=X
   NREG=.FALSE.
   IF(IDESC.NE.0.AND.IL1.LT.IFUNCT.AND.IL1.LT.IDESC.OR.
  C   IDESC.EQ.0.AND.IL1.LT.IFUNCT) GO TO 27
   DO 50 I=1,IMAX
   IF( .NOT.LOUT(I) ) GO TO 50
   IF( ISTATT(I).EQ.X.AND.IVECT(I).EQ.X ) GO TO 50
   LSAME = ( (     LOUTP(I)).AND.(     LPHAS1(I)).OR.
  C          (.NOT.LOUTP(I)).AND.(.NOT.LPHAS1(I)) )
   IMESS=40
   IF(ISTATT(I).EQ.L.AND.IVECT(I).EQ.L.AND.(.NOT.LSAME)) IMESS=41
```

continued

```
     IF(ISTATT(I).EQ.H.AND.IVECT(I).EQ.H.AND.(.NOT.LSAME)) IMESS=42
     IF(ISTATT(I).EQ.L.AND.IVECT(I).EQ.H.AND.(     LSAME)) IMESS=42
     IF(ISTATT(I).EQ.H.AND.IVECT(I).EQ.L.AND.(     LSAME)) IMESS=41
     IF( (     LENABL(I)).AND.IVECT(I).EQ.Z )               IMESS=43
     IF( (.NOT.LENABL(I)).AND.(LOUT(I)).AND.IVECT(I).NE.Z) IMESS=44
     IF(IMESS.NE.40) LERR=.TRUE.
     IF(IMESS.EQ.41) WRITE(1,41) NVECT,(ISYM1(J,I),J=1,8)
  41 FORMAT(/,' FUNCTION TABLE ERROR AT VECTOR',I3,' PIN =',8A1,
   C          '  EXPECT = H  ACTUAL = L')
     IF(IMESS.EQ.42) WRITE(1,42) NVECT,(ISYM1(J,I),J=1,8)
  42 FORMAT(/,' FUNCTION TABLE ERROR AT VECTOR',I3,' PIN =',8A1,
   C          '  EXPECT = L  ACTUAL = H')
     IF(IMESS.EQ.43) WRITE(1,43) NVECT,(ISYM1(J,I),J=1,8)
  43 FORMAT(/,' FUNCTION TABLE ERROR AT VECTOR',I3,'PIN =',8A1,
   C       /,'  EXPECT  = OUTPUT ENABLE ACTUAL = Z')
     IF(IMESS.EQ.44) WRITE(1,44) NVECT,(ISYM1(J,I),J=1,8),IVECT(I)
  44 FORMAT(/,' FUNCTION TABLE ERROR AT VECTOR',I3,'PIN =',8A1,
   C          '  EXPECT = Z ACTUAL = ',A1)
  50 CONTINUE
     DO 65 I=1,20
        DO 55 J=1,IMAX
        IF(IPIN(J).NE.I) GO TO 55
        IF( IVECT(J).EQ.L.OR.IVECT(J).EQ.H ) GO TO 51
        ISTATE(I)=IVECT(J)
        GO TO 65
  51    LSAME=( (     LPHASE(I)).AND.(     LPHAS1(J)).OR.
   C            (.NOT.LPHASE(I)).AND.(.NOT.LPHAS1(J)) )
        IF( INOO.EQ.N1.AND.(I.EQ.15.OR.I.EQ.16) )  LOUT(J)=.TRUE.
        IF( (.NOT.LOUT(J)).AND.(     LSAME).AND.
   C          IVECT(J).EQ.L )                       ISTATE(I)=N0
        IF( (.NOT.LOUT(J)).AND.(     LSAME).AND.
   C          IVECT(J).EQ.H )                       ISTATE(I)=N1
        IF( (.NOT.LOUT(J)).AND.(.NOT.LSAME).AND.
   C          IVECT(J).EQ.L )                       ISTATE(I)=N1
        IF( (.NOT.LOUT(J)).AND.(.NOT.LSAME).AND.
   C          IVECT(J).EQ.H )                       ISTATE(I)=N0
        IF( (     LOUT(J)).AND.(     LSAME).AND.
   C          IVECT(J).EQ.L.AND.(     LENABL(J)) ) ISTATE(I)=L
        IF( (     LOUT(J)).AND.(     LSAME).AND.
   C          IVECT(J).EQ.H.AND.(     LENABL(J)) ) ISTATE(I)=H
        IF( (     LOUT(J)).AND.(.NOT.LSAME).AND.
   C          IVECT(J).EQ.L.AND.(     LENABL(J)) ) ISTATE(I)=H
        IF( (     LOUT(J)).AND.(.NOT.LSAME).AND.
   C          IVECT(J).EQ.H.AND.(     LENABL(J)) ) ISTATE(I)=L
        GO TO 65
  55 CONTINUE
  65 IF( (LCLOCK).AND.IVECT(J).NE.Z ) IVECTP(J)=IVECT(J)
     ISTATE(10)=X
     ISTATE(20)=N1
     WRITE(1,60) NVECT,(ISTATE(I),I=1,20)
  60 FORMAT(' ',I2,' ',20A1)
     GO TO 90
  95 IF(.NOT.LERR) WRITE(1,67)
  67 FORMAT(/,' PASS SIMULATION')
     RETURN
 100 ILERR=ILL+4
     WRITE(1,101) (IBUF(I,1),I=1,8),ILERR,(IPAGE(I,ILL),I=1,79)
 101 FORMAT(/,' ERROR SYMBOL = ',8A1,'      IN LINE NUMBER ',I3,
   C       /,' ',80A1,/,' THIS PIN NAME IS NOT DEFINED IN THE',
   C                    ' FUNCTION TABLE PIN LIST')
       RETURN
       END
C
C***********************************************************************
C
      SUBROUTINE FIXTST(LPHAS1,LBUF,IC1,IL1,ISYM,ISYM1,IBUF,
     C                 IVECT,IVECTP,ITEST,LCLOCK,NREG,LFIX)
      BYTE     ISYM(8,20),ISYM1(8,20),IBUF(8,20),IVECT(20),IVECTP(20),
     C         IPAGE,L,H,X,Z
      LOGICAL  LBLANK,LLEFT,LAND,LOR,LSLASH,LEQUAL,LRIGHT,LXOR,LXNOR,
     C         LFIX,LPHAS1(20),LBUF(20),LCLOCK,NREG,TOR,TXOR,TXNOR,TAND,
     C         LPHASA,LPHASB
      COMMON   LBLANK,LLEFT,LAND,LOR,LSLASH,LEQUAL,LRIGHT,LXOR,LXNOR
      COMMON /PGE/ IPAGE(80,100)
      DATA L/'L'/,H/'H'/,X/'X'/,Z/'Z'/
      CALL GETSYM(LBUF,IBUF,1,IC1,IL1,LFIX)
      CALL MATCH(IINP,IBUF,ISYM1)
      ITESTA=IVECT(IINP)
      LPHASA = ( (      LBUF(1)).AND.(      LPHAS1(IINP)).OR.
     C             (.NOT.LBUF(1)).AND.(.NOT.LPHAS1(IINP)) )
      IF( (.NOT.LCLOCK).OR.(NREG) ) GO TO 5
      CALL MATCH(IIFB,IBUF,ISYM)
      IF( IIFB.EQ.14.OR.IIFB.EQ.15.OR.IIFB.EQ.16.OR.IIFB.EQ.17 )
     C    ITESTA=IVECTP(IINP)
    5 IF( (.NOT.LPHASA).AND.ITESTA.EQ.L ) GO TO 10
      IF( (.NOT.LPHASA).AND.ITESTA.EQ.H ) GO TO 15
      GO TO 20
   10 ITESTA=H
      GO TO 20
   15 ITESTA=L
   20 IF( .NOT.LRIGHT ) GO TO 25
           ITEST=ITESTA
           RETURN
   25 TOR   = (LOR.AND.(.NOT.LXOR))
      TXOR  = (LXOR)
      TXNOR = (LXNOR)
      TAND  = (LAND.AND.(.NOT.LXNOR))
      CALL GETSYM(LBUF,IBUF,1,IC1,IL1,LFIX)
      CALL MATCH(IINP,IBUF,ISYM1)
      ITESTB=IVECT(IINP)
      LPHASB = ( (      LBUF(1)).AND.(      LPHAS1(IINP)).OR.
     C             (.NOT.LBUF(1)).AND.(.NOT.LPHAS1(IINP)) )
      IF( (.NOT.LPHASB).AND.ITESTB.EQ.L ) GO TO 30
      IF( (.NOT.LPHASB).AND.ITESTB.EQ.H ) GO TO 35
      GO TO 40
   30 ITESTB=H
      GO TO 40
   35 ITESTB=L
   40 ITEST=L
      IF(   (TOR).AND.(ITESTA.EQ.H.OR. ITESTB.EQ.H) )      ITEST=H
      IF(  (TXOR).AND.((ITESTA.EQ.H.AND.ITESTB.NE.H).OR.
     C                 (ITESTA.NE.H.AND.ITESTB.EQ.H) ))    ITEST=H
      IF( (TXNOR).AND.((ITESTA.EQ.ITESTB).OR.
     C                 (ITESTA.EQ.X.OR.ITESTB.EQ.X) ))     ITEST=H
      IF(  (TAND).AND.(ITESTA.NE.L.AND.ITESTB.NE.L) )      ITEST=H
      IF( (ITESTA.EQ.X.OR.ITESTA.EQ.Z).AND.(ITESTB.EQ.X) ) ITEST=X
      RETURN
      END
C
C************************************************************************
C
```

TESTAS1.PAL
Contributed by: Trevor G. Marshall
"PALs Simplify Complex Circuits," by Trevor G. Marshall. January, page 247.

```
PAL16L8
DPORT20                           ; These 3 lines may be used
DMA SEQUENCER                     ; as you see fit
FIRST TRY AT A WORKING MODEL      ; then put the pin list
/MEMR /SELECT /MEMW /HLDA86 NC Q0 Q1 Q2 Q3 GND
NC /FC0 /JAMCNTR /HOLD86 /AS /WRITE /DMAEN NC /DBEN VCC

;and, after a blank line, the device equations
     IF (HLDA86)  AS = /Q3 * /Q2 *  Q1 * /Q0 ; 0010 is count=2
                     + /Q3 * /Q2 *  Q1 *  Q0 ; 0011 is count=3
                     + /Q3 *  Q2 * /Q1 * /Q0 ; 0100 is count=4
                     + /Q3 *  Q2 * /Q1 *  Q0 ; 0101 is count=5
                     + /Q3 *  Q2 *  Q1 * /Q0 ; 0110 is count=6
                     + /Q3 *  Q2 *  Q1 *  Q0 ; 0111 is count=7
                     +  Q3 * /Q2 * /Q1 * /Q0 ; 1000 is count=8
                     +  Q3 * /Q2 * /Q1 *  Q0 ; 1001 is count=9
```

continued

TESTAS2.PAS
Contributed by: Trevor G. Marshall
"PALs Simplify Complex Circuits," by Trevor G. Marshall. January, page 247.

```
PAL16L8
DPORT20                           ; These 3 lines may be used
DMA SEQUENCER                     ; as you see fit
SECOND TRY AT A WORKING MODEL     ; then put the pin list
/MEMR /SELECT /MEMW /HLDA86 NC Q0 Q1 Q2 Q3 GND
NC /FC0 /JAMCNTR /HOLD86 /AS /WRITE /DMAEN NC /DBEN VCC

;and, after a blank line, the device equations
     IF (HLDA86) AS = /Q3 *  Q1          ;0X1X is counts 2,3,6,7
                    + /Q3 *  Q2 * /Q1    ; 010X is counts 4,5
                    +  Q3 * /Q2 * /Q1    ; 100X is counts 8,9

FUNCTION TABLE
Q3 Q2 Q1 Q0 /HLDA86 /AS
-----------------------------------------------------------------
XXXX H  Z      ;Check that the ouput goes tristate
LLLL L  H      ;not asserted for 0
LLLH L  H      ;or for 1
LLHL L  L      ;but true from count of 2
LLHH L  L      ;through
LHLL L  L
LHLH L  L
LHHL L  L
LHHH L  L
HLLL L  L
HLLH L  L      ;9
HLHL L  H      ;deasserted at 10
HLHH L  H      ;and should not come back
HHLL L  H      ;through
HHLH L  H
HHHL L  H
HHHH L  H      ;15
-----------------------------------------------------------------
DESCRIPTION
This PAL implements the Address Strobe function of the DPORT20 dualport
controller PAL of the DSI-020 design.
```

TSTHOLD1.PAL
Contributed by: Trevor G. Marshall
"PALs Simplify Complex Circuits," by Trevor G. Marshall. January, page 247.

```
PAL16R4
IC2
(C) Copyright 1984,1985 Definicon Systems Inc.
Hold arbitration PAL for DSI-32 Rev B, TM, 12/5/84, first try
CLK /HLDA CTTL NC NC NC /HOLD86 /RFIO NC GND
/EN NC /HOLD /RFSHACK /HLDA86 /RFIOI /HOLD86I NC NC VCC

RFIOI := RFIO            ;Latch the asynchronous inputs, first refresh request
HOLD86I := HOLD86        ;and now the access request from the 8086

IF(VCC) HOLD = HOLD86I   ;immediately we get a request tell the CPU
             + RFIOI

;And then resolve the priorities, waiting for the HLDA before acknowledging
;First the higher priority, the 8086
HLDA86 := HLDA * HOLD86I * /RFIOI * RFSHACK ;Refresh about to end
        + HLDA * HOLD86I * /RFSHACK * /HLDA86 ;O/Ps Inactive, do 8086
        + HLDA * HOLD86I * HLDA86                  ;Latch the acknowledge
;Then the refresh acknowledge
RFSHACK := HLDA * RFIOI * /HOLD86I * HLDA86 ;HLDA86 active and about to go
away
        +  HLDA * RFIOI * /RFSHACK * /HOLD86I * /HLDA86 ;OK,unless rfsh
pending
        +  HLDA * RFIOI * RFSHACK                  ;Latch the acknowledge
```

```
FUNCTION TABLE
CLK /EN    /HOLD86 /RFIO
/RFIOI /HOLD86I   /HOLD   /HLDA    /HLDA86 /RFSHACK
-------------------------------------------------------------------------------
 CL  HH      HH H H XX ;clock everything inactive
 CL  HH      HH H H HH ;clock everything inactive
 CL  HL      LH L H HH ;RFIO recognized
 CL  HL      LH L L HL ;and acknowledged
 XL  HL      LH L L HL ;Check DIAGON function
 CL  HH      HH H H XX ;clock everything inactive
 CL  HH      HH H H HH ;clock everything inactive
 CL  LH      HL L H HH ;HOLD86 recognized
 CL  LH      HL L L LH ;and acknowledged
 XL  LH      HL L L LH ;Check DIAGON function
 CL  HH      HH H H XX ;clock everything inactive
 CL  HH      HH H H HH ;clock everything inactive
 CL  LL      LL L H HH ;both arrive at once
 CL  LL      LL L L LH ;8086 wins
 CL  HL      LH L L LH ;8086 goes away,hold active
 CL  HL      LH L L HL ;rfsh wins now
-------------------------------------------------------------------------------
DESCRIPTION
The HOLD PAL arbitrates between two possible sources of bus requests to
the 32032, refresh and PC-BUS access.
```

TSTHOLD2.PAL
Contributed by: Trevor G. Marshall
"PALs Simplify Complex Circuits," by Trevor G. Marshall. January, page 247.

```
PAL16R4
IC2
(C) Copyright 1984,1985 Definicon Systems Inc.
Hold arbitration PAL for DSI-32 Rev B, TM, 3/5/85, Implements T2 lockin
CLK /HLDA CTTL /TSO NC NC /HOLD86 /RFIO /ADS GND
/EN NC /HOLD /RFSHACK /HLDA86 /RFIOI /HOLD86I NC /T1 VCC

;To get around the problems in the 32032 we want to lockout holds except for
T2
;This is done with a single tristate output latch, using feedback
IF(/CTTL) T1 = ADS ;When ADS pulses and CTTL is low then /T1 will go low
             + T1  ;and latch low until CTTL goes high and releases it all

RFIOI := RFIO * T1       ;Latch the asynchronous inputs only when /T1 is true
       + RFIO * HOLD     ;If already holding, stay holding and ignore lockouts

HOLD86I := HOLD86 * T1  ;same for 8086 requests
        +  HOLD86 * HOLD

IF(VCC) HOLD = HOLD86I  ;immediately we get a request tell the CPU
             + RFIOI

;And then resolve the priorities, waiting for the HLDA before acknowledging
;First the higher priority, the 8086
HLDA86 := HLDA * HOLD86I * /RFIOI * RFSHACK ;Refresh about to end
        + HLDA * HOLD86I * /RFSHACK * /HLDA86 ;O/Ps Inactive, do 8086
        + HLDA * HOLD86I * HLDA86              ;Latch the acknowledge
;Then the refresh acknowledge
RFSHACK := HLDA * RFIOI * /HOLD86I * HLDA86 ;HLDA86 active and about to go
away
        +  HLDA * RFIOI * /RFSHACK * /HOLD86I * /HLDA86 ;OK,unless rfsh
pending
        +  HLDA * RFIOI * RFSHACK              ;Latch the acknowledge

FUNCTION TABLE
-------------------------------------------------------
-------------------------------------------------------
DESCRIPTION
The HOLD PAL arbitrates between two possible sources of bus requests to
the 32032, refresh and PC-BUS access. This version only applies HOLDS
to the CPU during T2.
```

continued

TSTHOLD3.PAL
Contributed by: Trevor G. Marshall
"PALs Simplify Complex Circuits," by Trevor G. Marshall. January, page 247.

```
PAL16R4
IC2
(C) Copyright 1984,1985 Definicon Systems Inc.
Hold arbitration PAL for DSI-32 Rev B, TM, 3/5/85, implements T4 lockin
CLK /HLDA CTTL /TSO NC NC /HOLD86 /RFIO /ADS GND
/EN NC /HOLD /RFSHACK /HLDA86 /RFIOI /HOLD86I NC /T1 VCC

;To get around the problems in the 32032 we want to lockout holds except for
T4
;This is done with a single tristate output latch, using feedback
IF(/CTTL) T1 = ADS ;When ADS pulses and CTTL is low then /T1 will go low
             + T1  ;and latch low until CTTL goes high and releases it all

RFIOI := RFIO*T1          ;recognize only during T1..BUT
       + RFIOI*TSO        ;Keep ready during TSO lockout
       + RFIO*HOLD        ;ok if 8086 is already holding for us

HOLD86I := HOLD86*T1
         + HOLD86*TSO
         + HOLD86*HOLD

IF(VCC) HOLD = HOLD86I*/TSO*/T1 ;Only assert HOLD if not in TSO or T2
             + RFIOI*/TSO*/T1
             + HOLD86I * HOLD   ;If HOLD already asserted by other source
             + RFIOI * HOLD     ; then ignore lockouts

;And then resolve the priorities, waiting for the HLDA before acknowledging
;First the higher priority, the 8086
HLDA86 := HLDA * HOLD86I * /RFIOI * RFSHACK ;Refresh about to end
        + HLDA * HOLD86I * /RFSHACK * /HLDA86 ;O/Ps Inactive, do 8086
        + HLDA * HOLD86I * HLDA86             ;Latch the acknowledge
;Then the refresh acknowledge
RFSHACK := HLDA * RFIOI * /HOLD86I * HLDA86 ;HLDA86 active and about to go
away
        +  HLDA * RFIOI * /RFSHACK * /HOLD86I * /HLDA86 ;OK,unless rfsh
pending
        +  HLDA * RFIOI * RFSHACK                 ;Latch the acknowledge

FUNCTION TABLE
--------------------------------------------------------
--------------------------------------------------------
DESCRIPTION
The HOLD PAL arbitrates between two possible sources of bus requests to
the 32032, refresh and PC-BUS access. This version only applies HOLDS
to the CPU during T2.
```

MODE.ASM
Contributed by: Chris H. Pappas and William H. Murray
"EGA Times 12," by Chris H. Pappas and William H. Murray. January, page 313.

```
;
;MODE.ASM, AN ASSEMBLY LANGUAGE PROGRAM WHICH WILL ALLOW
;SCREEN MODE CHANGES FROM THE SYSTEM LEVEL ON IBM PC/AT/XT
;AND COMPATIBLE SYSTEMS EQUIPPED WITH AN EGA BOARD.
;
;ENTER THE PROGRAM, ASSEMBLE, AND LINK TO OBTAIN THE .EXE FILE.
;
MYDATA      SEGMENT PARA 'DATA'
MESSAGE     DB      "Enter the screen mode desired (in Hexadecimal): $"
MYDATA      ENDS

MYCODE      SEGMENT PARA 'CODE      ;DEFINE CODE SEG. FOR MASM
MYPROC      PROC    FAR             ;PROCEDURE IS NAMED MYPROC
            ASSUME  CS:MYCODE,DS:MYDATA
            PUSH    DS              ;SAVE LOCATION OF DS REG.
            SUB     AX,AX           ;GET A ZERO IN AX
            PUSH    AX              ;SAVE ZERO ON STACK, TOO
```

```
         MOV     AX,MYDATA        ;GET DATA LOCATION IN AX
         MOV     DS,AX            ;PUT IT IN DS REGISTER

         LEA     DX,MESSAGE       ;PRINT MESSAGE TO SCREEN
         MOV     AH,9
         INT     21H

         MOV     DL,0
DIGIT:   MOV     AH,01            ;PREPARE TO READ KEY INPUT
         INT     21H
         SUB     AL,30H           ;CONVERT FROM ASCII TO HEX DIGIT
         CMP     AL,0             ;IS NUMBER >=0?
         JL      DOIT             ;IF LOWER, MAKE JUMP FOR MODE SWITCH
         CMP     AL,9             ;IS NUMBER <=9?
         JLE     ACCEPT           ;IF SO, NUMBER IN 0-9 RANGE - ACCEPT IT
         SUB     AL,7             ;IF NOT 0-9, IS IT A LETTER (A TO F)?
         CMP     AL,0             ;CHECK FOR "A"
         JL      DOIT             ;IF NOT, QUIT PROCESS
         CMP     AL,06H           ;CHECK FOR "F"?
         JG      DOIT             ;IF NOT, MAKE JUMP FOR MODE SWITCH
ACCEPT:  MOV     CL,04H           ;PREPARE TO ACCUMULATE UP TO TWO DIGITS
         SAL     DL,CL            ;ROTATE DL CONTENTS ONE "DIGIT" TO LEFT
         ADD     DL,AL            ;ADD IN NEW "DIGIT"
         JMP     DIGIT            ;ANOTHER DIGIT?

DOIT:    MOV     AL,DL            ;PREPARE TO SWITCH SCREEN MODES
         MOV     AH,0
         INT     10H              ;CALL INTERRUPT

         RET                      ;RETURN CONTROL TO DOS
MYPROC   ENDP                     ;END PROCEDURE NAMED MYPROC
MYCODE   ENDS                     ;END CODE SEGMENT NAMED MYCODE

         END                      ;END WHOLE PROGRAM
```

MODULA.LST
Contributed by: Paul A. Sand
"Three Modula 2 Programming Systems," by Paul A. Sand.
January, page 333. All the programs from this review.

```
) MODULE filewrite;
(* Write 64 Kbyte disk file -- Logitech version *)
FROM FileSystem IMPORT
    File, Lookup, Close, WriteNBytes;

FROM SYSTEM IMPORT
    ADR;

CONST
    CHUNKSIZE = 128;   (* size of chunks in bytes *)
    NCHUNKS = 512;     (* number of chunks to write *)

TYPE
    chunkarray = ARRAY [1..CHUNKSIZE] OF CHAR;

VAR
        chunk : chunkarray;              (* one chunk *)
        cf : FILE;                       (* chunk file variable *)
        i : CARDINAL;                    (* loop control variable *)
        fs : FileState;                  (* status from Create/Close *)
        name : ARRAY [0..30] OF CHAR;    (* file name *)

BEGIN
    FOR i := 1 TO CHUNKSIZE DO
        chunk[i] := CHR(ORD('1') + (i - 1) MOD 8)
    END;
    name := "C:\TEST.DAT";
        Lookup(cf, name, TRUE);
    FOR i := 1 TO NCHUNKS DO
        WriteNBytes(cf, ADR(chunk), CHUNKSIZE)
    END;
    Close(cf)
```

continued

```
END filewrite.
MODULE fileread;
(* 64 Kbyte file read -- LOGITECH version *)
FROM FileSystem IMPORT
    File, Lookup, Close, ReadBytes;

FROM SYSTEM IMPORT
    ADR;

CONST
    CHUNKSIZE = 128;   (* size of chunks in bytes *)
    NCHUNKS = 512;     (* number of chunks to write *)

TYPE
    chunkarray = ARRAY [1..CHUNKSIZE] OF CHAR;

VAR
        chunk : chunkarray;                      (* one chunk *)
        cf : FILE;                                       (* chunk file variable
*)
        i : CARDINAL;                            (* loop control variable *)
        junk : CARDINAL;                                 (* status from
ReadBytes *)
        fs : FileState;                          (* status from Open & Close *)
        name : ARRAY [0..30] OF CHAR;   (* file name *)

BEGIN
    name := "C:\TEST.DAT";
    Lookup(cf, name, TRUE);
    FOR i := 1 TO NCHUNKS DO
        ReadNBytes(cf, ADR(chunk), CHUNKSIZE, junk)
    END;
    Close(cf);
END fileread.
MODULE calculations;
(* Modula-2 program to perform a series of real *)
(* multiplications and divisions *)

FROM RealInOut IMPORT
    WriteReal;

FROM InOut IMPORT
    WriteString, WriteLn;

CONST
    MAX = 5000;         (* number of iterations *)

VAR
    a, b, c : REAL;     (* used in calculations *)
    i : CARDINAL;       (* loop control variable *)

BEGIN
    a := 2.71828;
    b := 3.14159;
    c := 1.0;
    FOR i := 1 TO MAX DO
        c := c * a;
        c := c * b;
        c := c / a;
        c := c / b
    END;
    WriteString('Error = ');
    WriteReal(c - 1.0, 10);
    WriteLn
END calculations.
MODULE sieve;

FROM InOut IMPORT
    WriteLn, WriteString, WriteCard;

CONST
    SIZE = 7000;
```

```
VAR
    flags : ARRAY [0..SIZE] OF BOOLEAN;
    i, prime, k, count, iter : CARDINAL;

BEGIN
    WriteString('10 iterations');
        WriteLn;
    FOR iter := 1 TO 10 DO
        count := 0;
        FOR i := 0 TO SIZE DO
            flags[i] := TRUE
        END;
        FOR i := 0 TO SIZE DO
            IF flags[i] THEN
                prime := i + i + 3;
                k := i + prime;
                WHILE k <= SIZE DO
                    flags[k] := FALSE;
                    k := k + prime
                END;
                INC(count)
            END
        END
    END;
    WriteCard(count, 1);
    WriteString(' primes');
    WriteLn
END sieve.
MODULE scout;
(* test character output to screen *)
FROM InOut IMPORT
        Write;
CONST
        MAX = 10000;    (* number of iterations *)
VAR
        i : CARDINAL;
BEGIN
        FOR i := 1 to MAX DO
                Write('a')
        END
END scount.
MODULE precision;
(* determine storage req. and precision of REAL type *)
(* find smallest number for which (1.0 + eps) > 1.0 *)
FROM SYSTEM IMPORT
        TSIZE;
FROM InOut IMPORT
        WriteString, WriteCard, WriteLn;
FROM RealInOut IMPORT
        WriteReal;
VAR
        eps : REAL;
BEGIN
        WriteString('Size of REAL = ');
        WriteCard(TSIZE(REAL), 1);
        WriteString(' bytes');
        WriteLn;
        eps := 1.0;
        REPEAT
                WriteReal(eps, 10);
                WriteLn;
                eps := eps/2.0
        UNTIL 1.0 + eps = 1.0;
END precision.
MODULE underflow;
(* find smallest positive REAL *)
FROM InOut IMPORT
        WriteLn;
FROM RealInOut IMPORT
        WriteReal;
VAR
        x : REAL;
BEGIN
        x := 1.0;
        REPEAT
```

continued

```
                WriteReal(x, 10);
                WriteLn;
                x := x/2.0
        UNTIL x = 0.0
END underflow.
MODULE overflow;
(* find largest positive REAL *)
FROM InOut IMPORT
        WriteLn;
FROM RealInOut IMPORT
        WriteReal;
VAR
        x : REAL;
BEGIN
        x := 1.0;
        REPEAT
                WriteReal(x, 10);
                WriteLn;
                x := 2.0 * x
        UNTIL FALSE
END overflow.

MODULE Dhrystone;
FROM InOut IMPORT
    WriteLn, WriteInt, WriteString;

FROM RealInOut IMPORT
    WriteReal;

FROM Storage IMPORT
    ALLOCATE, DEALLOCATE;

FROM Strings IMPORT
    CompareStr;
FROM TimeDate IMPORT
    GetTime, Time;

CONST
    NumberOfExecutions = 10000;
    NumberOfMeasurements = 10;
    LargeRealNumber = 1000000.0;
    MicrosecondsPerClock = 1000.0;
TYPE
    Enumeration = (Ident1, Ident2, Ident3, Ident4, Ident5);
    OneToThirty = [1..30];
    OneToFifty = [1..50];
    CapitalLetter = ['A'..'Z'];
    String30 = ARRAY[0..30] OF CHAR;
    Array1DimInteger = ARRAY OneToFifty OF INTEGER;
    Array2DimInteger = ARRAY OneToFifty, OneToFifty OF INTEGER;
    RecordPointer = POINTER TO RecordType;
    RecordType = RECORD
                PointerComp : RecordPointer;
                CASE Discr : Enumeration OF
                Ident1 :
                                EnumComp : Enumeration;
                                IntComp : OneToFifty;
                                StringComp : String30;
                |
                Ident2 :
                                EnumComp2 : Enumeration;
                                StringComp2 : String30;
                |
                Ident3, Ident4, Ident5 :
                                CharComp1, CharComp2 : CHAR;
                END;
       END;
VAR
    ExecutionIndex : [1..NumberOfExecutions];
    MeasurementIndex : [1..NumberOfMeasurements];
    BeginClock, EndClock, SumClocks, EmptyLoopClocks,
        TimePerExecution, SumTime, MinTime : REAL;
    PointerGlob, NextPointerGlob : RecordPointer;
    IntGlob : INTEGER;
    BoolGlob : BOOLEAN;
    CharGlob1, CharGlob2 : CHAR;
    ArrayGlob1 : Array1DimInteger;
    ArrayGlob2 : Array2DimInteger;
    IntGlob1, IntGlob2, IntGlob3 : OneToFifty;
    CharIndex : CHAR;
    EnumGlob : Enumeration;
    StringGlob1, StringGlob2 : String30;
PROCEDURE Proc1(PointerParVal : RecordPointer);
BEGIN
    WITH PointerParVal^.PointerComp^ DO
                PointerParVal^.PointerComp^ := PointerGlob^;
                PointerParVal^.IntComp := 5;
                IntComp := PointerParVal^.IntComp;
                PointerComp := PointerParVal^.PointerComp;
                Proc3(PointerComp);
                IF Discr = Ident1 THEN
                        IntComp := 6;
                Proc6(PointerParVal^.EnumComp, EnumComp);
                PointerComp := PointerGlob^.PointerComp;
                Proc7(IntComp, 10, IntComp);
                ELSE
                PointerParVal^ := PointerParVal^.PointerComp^;
                END;
    END;
END Proc1;
PROCEDURE Proc2(VAR IntParRef : OneToFifty);
VAR
    IntLoc : OneToFifty;
    EnumLoc : Enumeration;
BEGIN
    IntLoc := IntParRef + 10;
    REPEAT
                IF CharGlob1 = 'A' THEN
                IntLoc := IntLoc - 1;
                IntParRef := IntLoc - CARDINAL(IntGlob);
                EnumLoc := Ident1;
                END;
    UNTIL EnumLoc = Ident1;
END Proc2;
PROCEDURE Proc3(VAR PointerParRef : RecordPointer);
BEGIN
    IF PointerGlob <> NIL THEN
                PointerParRef := PointerGlob^.PointerComp;
    ELSE
                IntGlob := 100;
    END;
    Proc7(10, IntGlob, PointerGlob^.IntComp);
END Proc3;
PROCEDURE Proc4();
VAR
    BoolLoc : BOOLEAN;

BEGIN
    BoolLoc := CharGlob1 = 'A';
    BoolLoc := BoolLoc OR BoolGlob;
    CharGlob2 := 'B';
END Proc4;
PROCEDURE Proc5();
BEGIN
    CharGlob1 := 'A';
    BoolGlob := FALSE;
END Proc5;
PROCEDURE Proc6(EnumParVal : Enumeration; VAR EnumParRef : Enumeration);
BEGIN
    EnumParRef := EnumParVal;
    IF NOT Func3(EnumParVal) THEN
                EnumParRef := Ident4;
    END;
    CASE EnumParVal OF
                Ident1 :
                EnumParRef := Ident1;
                |
                Ident2 :
                IF IntGlob > 100 THEN
                                EnumParRef := Ident1;
```

continued

```
                ELSE
                                EnumParRef := Ident4;
                END;
                |
                Ident3 :
                        EnumParRef := Ident2;
                |
                Ident4 :
                Ident5 :
                EnumParRef := Ident3;
        END;
END Proc6;
PROCEDURE Proc7(IntPar1Val, IntPar2Val : OneToFifty;
    VAR IntParRef : OneToFifty);
VAR
    IntLoc : OneToFifty;

BEGIN
    IntLoc := IntPar1Val + 2;
    IntParRef := IntPar2Val + IntLoc;
END Proc7;
PROCEDURE Proc8(VAR ArrayPar1Ref : Array1DimInteger;
        VAR ArrayPar2Ref : Array2DimInteger;
        IntPar1Val, IntPar2Val : INTEGER);
VAR
    IntIndex, IntLoc : OneToFifty;
BEGIN
    IntLoc := IntPar1Val + 5;
    ArrayPar1Ref[IntLoc] := IntPar2Val;
    ArrayPar1Ref[IntLoc + 1] := ArrayPar1Ref[IntLoc];
    ArrayPar1Ref[IntLoc + 30] := IntLoc;
    FOR IntIndex := IntLoc TO IntLoc + 1 DO
                ArrayPar2Ref[IntLoc, IntIndex] := IntLoc;
    END;
    ArrayPar2Ref[IntLoc, IntLoc - 1] := ArrayPar2Ref[IntLoc,IntLoc - 1] + 1;
    ArrayPar2Ref[IntLoc + 20,IntLoc] := ArrayPar1Ref[IntLoc];
    IntGlob := 5;
END Proc8;
PROCEDURE Func1(CharPar1Val, CharPar2Val : CapitalLetter) : Enumeration;
VAR
    CharLoc1, CharLoc2 : CapitalLetter;
BEGIN
    CharLoc1 := CharPar1Val;
    CharLoc2 := CharLoc1;
    IF CharLoc2 <> CharPar2Val THEN
                RETURN Ident1;
    ELSE
                RETURN Ident2;
    END;
END Func1;
PROCEDURE Func2(VAR StringPar1Ref, StringPar2Ref : String30) : BOOLEAN;
VAR
    IntLoc : OneToThirty;
    CharLoc : CapitalLetter;
BEGIN
    IntLoc := 2;
    WHILE IntLoc <= 2 DO
                IF Func1(StringPar1Ref[IntLoc], StringPar2Ref[IntLoc+1]) =
Ident1 THEN
                CharLoc := 'A';
                IntLoc := IntLoc + 1;
                END;
    END;
    IF (CharLoc >= 'W') AND (CharLoc < 'Z') THEN
                IntLoc := 7;
    END;
    IF CharLoc = 'X' THEN
                RETURN TRUE;
    ELSIF CompareStr(StringPar1Ref, StringPar2Ref) > 0 THEN
                IntLoc := IntLoc+7;
                RETURN TRUE;
    ELSE
                RETURN FALSE;
    END;
```

```
END Func2;
PROCEDURE Func3(EnumParVal : Enumeration) : BOOLEAN;
VAR
    EnumLoc : Enumeration;

BEGIN
    EnumLoc := EnumParVal;
    IF EnumLoc = Ident3 THEN
                RETURN TRUE;
    END;
END Func3;
PROCEDURE clock() : REAL;
VAR
    Now: Time;
    milliseconds: CARDINAL;
    seconds: CARDINAL;
    minutes: CARDINAL;
    hours: CARDINAL;

BEGIN
    GetTime(Now);
    WITH Now DO
                seconds := millisec DIV 1000;
                milliseconds := millisec MOD 1000;
                hours := minute DIV 60;
                minutes := minute MOD 60
    END;
    RETURN FLOAT(milliseconds)+1000.0*(FLOAT(seconds)+60.0*(FLOAT(
                minutes)+60.0*FLOAT(hours)));
END clock;
BEGIN
    NEW(NextPointerGlob);
    NEW(PointerGlob);
    PointerGlob^.PointerComp := NextPointerGlob;
    PointerGlob^.Discr := Ident1;
    PointerGlob^.EnumComp := Ident3;
    PointerGlob^.IntComp := 40;
    PointerGlob^.StringComp := 'DHRYSTONE PROGRAM, SOME STRING';
    StringGlob1 := "DHRYSTONE PROGRAM, 1'ST STRING";
    WriteLn();
    WriteString('Dhrystone Benchmark (March 84), Version Pascal / 2');
    WriteLn();
    WriteString('Times are CPU user time per execution, in microseconds');
    WriteLn();
    WriteLn();
    SumTime := 0.0;
    MinTime := LargeRealNumber;
    FOR MeasurementIndex := 1 TO NumberOfMeasurements DO
                BeginClock := clock();
                Array2Glob[8][7] := 10;
                FOR ExecutionIndex := 1 TO NumberOfExecutions DO
                Proc5();
                Proc4();
                        IntGlob1 := 2;
                IntGlob2 := 3;
                StringGlob2 := "DHRYSTONE PROGRAM, 2'ND STRING";
                EnumGlob := Ident2;
                BoolGlob := Func2(StringGlob1,StringGlob2);
                WHILE IntGlob1<IntGlob2 DO
                                IntGlob3 := 5 * IntGlob1 - IntGlob2;
                                Proc7(IntGlob1, IntGlob2, IntGlob3);
                                IntGlob1 := IntGlob1+1;
                END;
                Proc8(ArrayGlob1, ArrayGlob2, IntGlob1, IntGlob3);
                        Proc1(PointerGlob);
                FOR CharIndex := 'A' TO CharGlob2 DO
                                IF EnumGlob = Func1(CharIndex, 'C') THEN
                                         Proc6(Ident1, EnumGlob);
                                END;
                END;
                IntGlob3 := IntGlob2 * IntGlob1;
                IntGlob2 := IntGlob3 DIV IntGlob1;
                IntGlob2 := 7 * (IntGlob3 - IntGlob2) - IntGlob1;
                Proc2(IntGlob1);
                END;
```

continued

```
EndClock := clock();
SumClocks := (EndClock - BeginClock) * MicrosecondsPerClock;
BeginClock := clock();
FOR ExecutionIndex := 1 TO NumberOfExecutions DO
END;
EndClock := clock();
                EmptyLoopClocks := (EndClock - BeginClock) *
MicrosecondsPerClock;
                SumClocks := SumClocks - EmptyLoopClocks;
                TimePerExecution := SumClocks / FLOAT(NumberOfExecutions);
                WriteString('Time for run ');
                WriteInt(MeasurementIndex, 4);
                WriteString(': ');
                WriteReal(TimePerExecution, 10);
                WriteLn();
                SumTime := SumTime+TimePerExecution;
                IF TimePerExecution<MinTime THEN
                        MinTime := TimePerExecution;
                END;
    END;
    WriteLn();
    WriteString('Average execution time: ');
    WriteReal(SumTime/FLOAT(NumberOfMeasurements), 10);
    WriteLn();
    WriteLn();
    WriteString('Minumum execution time: ');
    WriteReal(MinTime, 10);
    WriteLn();
    WriteLn();
END Dhrystone.
```

PARTS.LIS
Contributed by: Robert A. Freedman
"A PAL Programmer," by Robert A. Freedman. January, page 263.

```
       Printed Circuit Board, WW     1     28.00     28.00 JDR MicroDevices

       Socket Module P. C. Board     1      8.00      8.00

       24 Pin ZIF Socket             1     15.80     15.80 3M-Textool

       20 Pin ZIF Socket             1     13.27     13.27

       RS-232 D-Sub 25-S Rt. Ang.    1      3.29      3.29 R. S. Cat # 276-
1521

       RS-232 D-Sub 25-P             1      3.29      3.29

       UNC5810A Sprague              3      2.80      8.40

       UNC5821A                      4      2.70     10.80

       UNC5895A                      1      2.34      2.34

       IRFD-9123 HEXDIP Power FET    1      2.25      2.25

       7406                          1      0.40      0.40

       LS138                         1      0.49      0.49

       LS245                         1      0.99      0.99

       LS251                         2      0.50      1.00

       LS259                         1      1.19      1.19

       LS273                         2      0.79      1.58

       LS390                         1      0.89      0.89

       PAL 16L8                      2      3.00      6.00

       PAL 16R8                      1      3.00      3.00
```

	DAC-08 EP	2	2.09	4.18	
	LM-317 T-220, Adjust. Reg.	2	1.89	3.78	
	LM-324, Quad OP-Amp	1	0.40	0.40	
	LM-336, 2.5 V Reference	1	0.40	0.40	
	LM-339, Quad Comparator	3	0.40	1.19	
	TL-497ANC	1	1.49	1.49	
	1N4002 Diode	6	0.20	1.20	
	1N4740A, 10 Volt Zener	1	0.25	0.25	
	1N4935 Fast Recov. Diode	1	0.25	0.25	
	100. Ohm 1/4 watt 5% Res.	1	0.00	0.00	
	240. Ohm 1/4 watt 5% Res.	2	0.00	0.00	
	1.0K Ohm 1/4 watt 5% Res.	1	0.05	0.05	
	1.2K Ohm 1/4 watt 5% Res.	2	0.05	0.10	
	2.0K Ohm 1/4 watt 5% Res.	1	0.05	0.05	
	2.2K Ohm 1/4 watt 5% Res.	1	0.05	0.05	
	2.7K Ohm 1/4 watt 5% Res.	1	0.05	0.05	
OK	5.1K Ohm 1/4 watt 5% Res.	8	0.05	0.40	1% better, but 5%
	5.6K Ohm 1/4 watt 5% Res.	1	0.05	0.05	
	15.K Ohm 1/4 watt 5% Res.	1	0.05	0.05	
	8-pin IC Sockets	1	0.20	0.20	
	14-pin IC Sockets	8	0.28	2.24	
	16-pin IC Sockets	16	0.32	5.12	
	18-pin IC Sockets	3	0.36	1.08	
	20-pin IC Sockets	7	0.40	2.80	
	24-pin IC Sockets	1	1.00	1.00	
	DALE IHA-203 100uH	1	4.00	4.00	
	or any 100-250 uH @ 1 Amp	0	0.00	0.00	
	100 pf Mica Cap	1	0.33	0.33	
	.01 Mfd Monolythic Caps	2	0.15	0.30	
	0.1 Mfd Monolythic Caps	40	0.15	6.00	
	15 Mfd @ 20V Tantalum Cap	4	0.50	2.00	
	22 Mfd @ 25V Tantalum Cap	2	1.00	2.00	
	470 Mfd @ 16V Aluminum Cap	1	1.00	1.00	
	1.0 Ohm 1 Watt Resistor	1	0.79	0.79	
	Resistor Sip 1.0K x 7	1	0.35	0.35	
	Resistor Sip 2.2K x 9	3	0.35	1.05	
	Resistor Sip 4.7K x 7	2	0.35	0.70	
	Resistor Sip 4.7K x 5	2	0.35	0.70	

ZAPAL.C
Contributed by: Robert A. Freedman
"A PAL Programmer," by Robert A. Freedman. January, page 263.

```
/*      ZAPAL.C - Byte Magazine ZAP-A-PAL Programmer for IBM-PC         */

/*      Version 1.1 - by Robert A. Freedman -  2 Oct 1986 - 11:25 PM    */

#include <stdio.h>
/*     #include "stdio.h"       */
#define uchar unsigned char
#define ERROR -1
#define base  0x100
#define DAC_A base+0
#define DAC_B base+1
#define SCLK  base+2

#define STROBE base+0x8
#define ENAB   base+0x9         /* Enable BiMOS drivers */
#define ENCH   base+0xA
#define ENCL   base+0xB
#define VLH    base+0xD
#define VINHIB base+0xE
#define TRIG   base+0xF

#define BUSY            inportb(base+0xC) & 1
#define CAL_REF_10V     inportb(base+0xA) & 1
#define CAL_REF_2P5     inportb(base+0xB) & 1
#define P23 base+0x8
#define P22 base+0x0
#define P21 base+0x1
#define P20 base+0x2
#define P19 base+0x3
#define P18 base+0x4
#define P17 base+0x5
#define P16 base+0x6
#define P15 base+0x7
#define P14 base+0x9

int c,e,err,k,from,to,busy,vcc,vihh,v5_0,v5_5,v6_0,v11_75;
int vcc_want,vihh_want;

static int vcclo,vcchi,vihlo,vihhi;     /* Auto-Cal Points       */
       int dac;

static int verpin,vad,fuse; /* Pin # to verify, I/O adr, State  */
static int veradr[10] =
     {9,7,6,5,4,3,2,1,0,8};  /* Mux adr for Pins 14 - 23 */

uchar pins[32];
/*             Outputs         Control               Inputs
        2 2 2 1 1 1 1 1 2 1    1                             1 1
        2 1 0 9 8 7 6 5 3 4  1 3             2 3 4 5 6 7 8 9 0 1     */
static uchar clear[28] =
    {0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0}; /* Clear*/
static uchar odlo[28] =
    {0,0,0,0,0,0,0,0,0,0, 2,1,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0}; /* OD lo*/
static uchar odhi[28] =
    {0,0,0,0,0,0,0,0,0,0, 1,2,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0}; /* OD hi*/
static uchar x1[28] =
    {0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0};
static uchar x2[28] =
    {0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0};
static uchar x3[28] =
    {0,1,2,1,2,1,4,1,4,1, 0,1,0,1,0,1,0,1, 0,1,0,1,0,1,0,1,0,1};

static int pind[24] = /* Maps Pin #'s to Shift Register Position  */
    {10,18,19,20,21,22,23,24,25,26,27,28,11,9,7,6,5,4,3,2,1,0,8};

static s;       /* s is Style of fuze-map print-out      */

static char *chip;      /* Point to string ie "16L8"     */
```

```
int ci,chip_index;
static char chip_nam[32][10] =  /* Array of "Chip Names" */
        { "16L8","16R8","16R6","16R4","20L8","20R8","20R6","20R4" };

static uchar npins[32] =      /* either 20 or 24, 0 = empty   */
        { 20,20,20,20, 24,24,24,24 };


unsigned seg,offset,off,word;
struct regval { int ax,bx,cx,dx,si,di,ds,es; };
struct regval *sreg,*rreg;

char rbuf[128];                 /* buffer line from file        */

unsigned char byte,best;
unsigned int    cnt,fdi,fdo;

int     n,n_wr,i,j,sum,lr,ix,ino,af,T20,T24,fuzno;
char    *fn,filename[64],*p,fin[32],fon[32];

extern FILE *fopen();
FILE    *fo,*fp;

static int fuzmap[4096] =
     { 0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1 };

main(argc,argv) int argc; char **argv;
{       int j,fooz;

        printf ("\nZAP-A-PAL JEDEC Driver");
        strcpy(fin,"file.jed");
        strcpy(fon,"fmap.jed");

        T20 = 1;        /* Do 20-pin PALs       */
        ci = 0;

        vcc  = 131;     /* DAC-A 131 = 5.02 Volts       */
        vihh = 154;     /* DAC-B 154 = 11.75 Volts: 13.00 @ VIHH */

        menu();         shutdn();       exit(0);

        outportb(VINHIB,1);     /* Turn ON 497 Booster  */
        outportb(DAC_A,156);    /* Load DAC-A 156=6.0 V */
        outportb(DAC_B,0);      /* Load DAC-B  =  0 V   */

/*      if (argc < 2) erra("Use FIXUM outfile");        */

/*      strcpy(filename,argv[1]); */   /* Capture output filename       */

/*      fn = filename;  printf("\nfilename = %s\n",fn); */

/*      fp = fopen(fn,"wb"); if(!fp) erra("ERROR: opening");     */


/*      for (i=from;i<to;i++)   { byte = peek(i,0xC800) & 0xFF; };
        n_wr = fwrite(buf,1,cnt=n=8192,fp);     */

/*              if(n_wr != cnt) erra("write error");
                fclose(fp);
*/
}
menu()
{
        wipe();
        while ( 1 ) {

        crt_srcp(0,0,0);        printf(
"\tZAP-A-PAL Programmer - (C) 1986 by R. A. Freedman S# 00001");
        T20 = (npins[ci]==20?1:0);
        printf("\n\n\t\tPAL-%s has %2d pins and %4d fuzes\n"
                                ,chip_nam[ci],npins[ci],(T20?2048:3200) );

        printf("\n\tT - Select Device Type:");
        printf("\n\tL - Load PAL chip into RAM");
```

continued

```
        printf("\n\tR - Show fuze map from PAL Chip");
        printf("\n\tY - Show fuze map from RAM");
        printf("\n\tC - Compare PAL chip with RAM");
        printf("\n\tZ - Burn a PAL from RAM");
        printf("\n\tI - Set to READ  file %s",fin);
        printf("\n\tO - Set to WRITE file %s",fon);
        printf("\n\tJ - Load  JEDEC file into RAM *");
        printf("\n\tW - Write JEDEC file from RAM");
        printf("\n\tS - Style of Fuze MAP is %s ",(s&1?"JEDEC":"PALASM"));
        printf("\n\tU - FUZEs are shown as %s",(s&2?"'1' or '0'":"'X' or '-
'"));
        printf("\n\tG - %s GAP every 4th fuse",(s&4?"Insert":"Don't "));
        printf("\n\tM - Calibrate manually");
        printf("\n\tA - Auto-Calibrate");
        printf("\n\tQ - Quit to DOS \n\n\t");
/*      printf("\n\t? - more to come\n\n\t");   */

                c = tolower(bdos(1) & 0xFF);

                switch(c) {

                case 'm': calib();      break;
                case 'a': autocal();    break;
                case 'q': return(0);    break;
                case 'r': shopal(1,s);  break;  /* Show Fuz-Map from chip */
                case 'y': shopal(0,s);  break;  /* show Fuz-Map from RAM  */
                case 'z': zapal();      break;
                case 'l': loadpal();    break;
                case 'j': if(loadjedec()==ERROR){
                                printf("ERROR"); getchar(); }; break;
                case 'w':
                case 'p': paltype();    break;
                case 'c': shopal(2,s);  break;  /* Compare PAL with RAM */
                case 's': s = s ^ 1;    break;
                case 'u': s = s ^ 2;    break;
                case 'g': s = s ^ 4;    break;
                case 't': ci= (ci+1) & 7;       break;
                case 'i': getfn(fin);   break;
                case 'o': getfn(fon);   break;
                case 's': s = s ^ 1;    break;
                case 's': s = s ^ 1;    break;
                case 's': s = s ^ 1;    break;
                case 's': s = s ^ 1;    break;

                default :               break;
                };

        };
}
getfn(fn) char *fn;
{char *p; int l;
                printf(" Enter File.nam ");
                fgets(fn,30,stdin); l=strlen(fn);
                p = fn + l - 1;  *p = 0; /* zot \n      */
}
wipe()
{

printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
}
paltype(){}
loadjedec()     /* Load a JEDEC file into RAM   */
{
        fp = fopen(fin,"r");
                if (!fp) { printf("\nCan't OPEN %s",fin);
                                        getchar(); return(0); };
        for (fuzno=0; fuzno < (T20?2048:3200); fuzno++) {
                fuzmap[fuzno] = 2;      /* Mark as Empty        */
        };

        while (1) {
                while ( (c=fgetc(fp)) != '*') { if (c==ERROR) return(ERROR);
};
                c = fgetc(fp);  /* get command letter, Q G F L C #      */
                if (c == 'L') {
                        fuzno = 0; for (i=0;i<4;i++) { c = fgets(fp);
                                fuzno = fuzno * 10 + ani(c);    };
                        fgets(fp); /* Skip space        */

                        do { c = fgets(fp); if (c == ERROR) return(ERROR);
                                fuzmap[fuzno] = c & 1; fuzno++;
                        } while( c == '1' || c == '0') ;
                };
                if (c == 'C') { fclose(fp); return(0); };
                if (c == 'F') { };
        };
}
char bcda(p) char *p; {return(ani(*p++)*16+ani(*p++)); }
int ani(c) char c; {return( c >= 'A' ? (c-'7') : (c-'0') ); }

loadpal()       /* Load from PAL chip into RAM  */
{
        mount();
        for (fuzno=0; fuzno < (T20?2048:3200); fuzno++) {
                fuzmap[fuzno] = readfuz( fuzno );
        }; shutdn();
}
mount()
{       printf("\nPlease mount PAL in Zif Socket, Watch Pin 1");

                getchar();
                revup();        /* Turn ON the Power    */
}
shopal(f,s) int f,s;
/*        f = source:   0 = from RAM
                        1 = from PAL chip
                        2 = show differences

          s =   Style:  0
                        1 = JEDEC vs PALASM numbering
          bits          2 = "1" & '0'vs 'X' & '-'
                        4 = Space every 4 bits
                        8                                       */

{       int fooz,fuz,j;

        if(f != 0) mount(); s = s & 7;

        fuzno=0;  while ( fuzno < (T20?2048:3200) ) {

        /*      printf("\n-----------------------------" );     */

                printf ( (s&1?"\n*L%04d ":"\n%4d "),
                                        (s&1?fuzno:fuzno/(T20?32:40) ) );
                for (j=0;j<(T20?32:40);j++) {
                f = f & 3;
                if(f == 0) { fooz = fuzmap[fuzno]; };
                if(f == 1) { fooz = readfuz(fuzno); };
                if(f == 2) { fooz = fuzmap[fuzno] ^ readfuz(fuzno); };

        printf("%c",(f==2?
                        (!fooz ? '.' : (fuzmap[fuzno] ? 'O':'U') ):
                        (s&2?(fooz?'0':'1'):(fooz?'X':'-') )
                    )
                );
                        fuzno++;
                        if ( !(fuzno % 4) ) printf((s&4?" ":""));
                };
        }; shutdn();    getchar(); wipe();

}
autocal()       /* Automatic Calibration of Voltage Generators  */
{       int d,test;

        outportb(DAC_A,0);      /* Load DAC-A = 0 V     */
        outportb(DAC_B,0);      /* Load DAC-B = 0 V     */

        printf("\nPlease Remove PAL from socket during Calibration");
                getchar();
```

continued

```
        revup();        /* Turn ON the Power    */

        outportb(DAC_A,0);      /* Load DAC-A = 0 V     */
        outportb(DAC_B,0);      /* Load DAC-B = 0 V     */

        test = 0; d = 0; dac = 0;

                vcclo = slew(0,0); if (vcclo == ERROR) printf("\n\nPUNT");

                vcchi = slew(0,1); if (vcclo == ERROR) printf("\n\nPUNT");

        outportb(DAC_A,0);      /* Load DAC-A = 0 V     */
        outportb(DAC_B,0);      /* Load DAC-B = 0 V     */

        dac = 0;printf("\n");

                vihlo = slew(1,0); if (vcclo == ERROR) printf("\n\nPUNT");

                vihhi = slew(1,1); if (vcclo == ERROR) printf("\n\nPUNT");

        for ( i=0; i<=400; i+=5) { vcc_want = i; vihh_want = i;

                 vcc = ( ( (vcchi - vcclo) * 2 *  vcc_want ) / ( 15 * 20 ) );

                vihh = ( ( (vihhi - vihlo) * 2 * vihh_want ) / ( 15 * 20 ) );

                printf ("\n %4d / 20 =>  vcc = %4d" ,vcc_want ,vcc);

                printf ("   %4d / 20 => vihh = %4d",vihh_want,vihh);
        };

}
int slew(d,test) int d,test;    /*      Scan for Thresholds     */
{
        do { if (dac > 255) return(ERROR);

                outportb((d?DAC_B:DAC_A),dac);    /* Load DAC-(A or B) */

                printf("\r%s = %3d @ 2.5Volt Point and %3d @ 10.0V Point %3d",
                (d?"vihh":" vcc"),(d?vihlo:vcclo),(d?vihhi:vcchi),dac++ );

           } while ( !(test? CAL_REF_10V : CAL_REF_2P5) );
        return(dac);
}

calib()         /* Calibrate Programmable Voltage Sources       */
{       int d,dac[2]; d = 0; dac[0] = 0; dac[1] = 0;

        printf("\nType 'C' to Calibrate, anything else to use defaults-");
        c = tolower( getchar() ); if (c != 'c' ) return(0);

        revup();        /* Turn ON the Power    */

        outportb(DAC_A,0);      /* Load DAC-A = 0 V     */
        outportb(DAC_B,0);      /* Load DAC-B = 0 V     */

        while (1) {

                outportb((d?DAC_B:DAC_A),dac[d]);    /* Load DAC-(A or B) */

                c = tolower( getchar() );
                if (c=='q' ) {vcc = dac[0]; vihh = dac[1]; break; };
                if (c=='v') d = ++d & 1;        /* Other DAC    */

                if (c=='u') dac[d] = ++dac[d] & 0xFF;
                if (c=='d') dac[d] = --dac[d] & 0xFF;

        printf ( "\rvcc=%4d, vihh=%4d %s %s",dac[0],dac[1],
                                        (CAL_REF_10V ? "High":" Low"),
                                        (CAL_REF_2P5 ? "High":" Low") );
        };
}
revup()         /* Set up DACs and Turn On Power        */
{
        outportb(ENAB,0);       /* Disable BiMOS drivers        */
        outportb(ENCL,1);       /* Disable BiMOS drivers CLOCK  */
        outportb(ENCH,1);       /* Disable BiMOS drivers OD     */
        outportb(VINHIB,1);     /* Turn ON 497 Booster  */
        outportb(VLH,0);        /* Set Booster to Low (15V)     */
        outportb(TRIG,0);       /* Make sure TRIG is low        */

        outportb(DAC_A,vcc);    /* Load DAC-A = vcc Volts       */

        outportb(DAC_B,vihh);   /* Load DAC-B = VIHH Volts      */

        printf ("\nvcc = %3d - vihh = %3d",vcc,vihh);
}
shutdn()
{
        outportb(ENAB,0);       /* Disable BiMOS drivers        */
        outportb(ENCL,1);       /* Disable BiMOS drivers CLOCK  */
        outportb(ENCH,1);       /* Disable BiMOS drivers OD     */

        outportb(VINHIB,0);     /* Turn OFF 497 Booster */

        outportb(DAC_A,0);      /* Load DAC-A  =  0 V   */

        outportb(DAC_B,0);      /* Load DAC-B  =  0 V   */

}
zapal()         /* Burn a PAL from RAM image    */
{       int e,x;
        mount();
        for (fuzno=0; fuzno < (T20?2048:3200); fuzno++) {

                if (fuzmap[fuzno] >= 2) continue;
                if (fuzmap[fuzno] == 0) continue;

                if (fuzmap[fuzno] == 1) {

                /* Set-up and see if fuze is already blown      */

                x = do_a_fuz(fuzno);

                /* for 16L8 fuze is low if blown        */

        /*      if (!x) continue; */    /* Skip if blown */

                        if( (e=zot()) == ERROR) {
                                printf("\nCan't Program Fuze # %4d ",fuzno);
                                getchar();      break; };
                };
        }; shutdn(); return(e);
}
int readfuz(fuzno) int fuzno;   /* Read state of selected fuse  */
{
        return( do_a_fuz(fuzno) );
}
int do_a_fuz(fuzno) int fuzno;  /* Set up to read or write a fuze */
{       int half,pln;
        pln = ( fuzno / ( T20 ? 32 : 40 ) );
        if (pln > (T20?63:79) ) return(ERROR);

        outportb(ENAB,0);       /* Disable BiMOS drivers        */
        outportb(ENCL,1);       /* Disable BiMOS drivers CLOCK  */
        outportb(ENCH,1);       /* Disable BiMOS drivers OD     */

        half = ( T20 ? 32 : 40 );       /* Set OD and CLOCK pins        */
        pin( 1, (pln >= half?1:2) );    pin(13, (pln >= half?2:1) );

        (pln >= half ? ldsr(odhi) : ldsr(odlo) );       /* Shift OD & Clock */

        outportb(ENCL,0);       /* Enable BiMOS drivers CLOCK   */
        outportb(ENCH,0);       /* Enable BiMOS drivers OD      */

        selfuz(fuzno);          ldsr(pins);

        outportb(ENAB,1);       /* Enable BiMOS drivers         */

        return( verifuz() );    /* Return state of addressed fuze       */
```

continued

```
}
zot()
{       int i;
        for (i=0;i<5-4;i++) {

                outportb(TRIG,1);
                outportb(TRIG,1);
                while ( !BUSY ) {};
                outportb(TRIG,0);
                while ( BUSY ) {};

                if ( verifuz() ) return(1);
        }; return(ERROR);

}
int verifuz()   /*      Return state of fuze    */
{
        outportb(ENCL,1);       /* Pulse CLOCK pin by floating  */
        outportb(ENCL,0);       /* CLOCK to Z momentarily       */

        vad = veradr[verpin-14] + base; /* Compute Mux adr of Pin       */
        fuse = inportb(vad) & 1;        /* Read the state of the fuse   */
        return(fuse);                   /* 0 = Blown, 1 = Intact fuse   */
}
selfuz(fuzno) int fuzno;
{       int an,half,of,ox,lino,pl,pln,i;

        ldsr(clear);    /* Clear out old fuze info      */

        /*      Compute and place input pins    */

        lino =( fuzno % ( T20 ? 32 : 40 ) );

        pln = ( fuzno / ( T20 ? 32 : 40 ) );
        if (pln > (T20?63:79) ) return(ERROR);

        lr = 0; if   (lino & 2)  lr = 2;        /* Find which half       */

        ix = 0; if (!(lino & 1)) ix = 1;        /* Find the state of Pin x */

        /* Now find where to put the selected pin, ie [I0..I9]          */

        for (i=0;i<10;i++) pin(2+i,2); /* Pull all input pins to VIHH   */

        ino = lino / 4; pin(2+ino,ix);  /* Then set Selected pin to TTL  */

        /*      Compute and Place Output Pins         */

        half = ( T20 ? 32 : 40 );       /* Set OD and CLOCK pins        */
        pin( 1, (pln >= half?1:2) );    pin(13, (pln >= half?2:1) );

        pl = pln; if(pln >= half) pl = pln-half;
        an =  pl % 8;                           /* A0..An = pl mod 8    */
        ox = (pl / 8) & (T20?0xF:0x1F);         /* Select Outp Pin to pulse */
        for (i=14;i<=23;i++) { pin(i,0); };     /* Clear all Outputs    */

                             af = (T20?16:15 ); of = (T20?22:23 );
        if ( pln >= half ) { af = (T20?19:19 ); of = (T20?18:18 ); };
        if ((pln <  half) && !T20 ) an = bitinv(an,4);
        an = an & (T20? 7 : 0xF );

        for ( i = (T20?2:3); i >= 0; i--) {  /* Set Address bits  */
                pin(af+i,( an % 2 ? 2 : 0 ) ); an = an / 2;
        };
        pin(of-ox,4);   /* Set Output Pin to Pulse      */
        verpin = of-ox; /* Save pin to verify fuse state        */
        pin( (pln < half ? (T20?15:14) : (T20?22:23) ),lr); /* Set L/R  */

}
int pin(n,val) int n,val;       /* Read or Store value of a pin */
{int v; uchar *p;       if (n == 0 || n > (T20?24:24) ) val = 0xE;
                        p = pins + *(pind + n - 1 );
                        v = *p; *p = val; return(v);
}
```

```
shopin()
{       printf("\n");
        for (i=0;i<28;i++) {     if( i == 10 || i == 18 ) printf(" -");
                     if( i ==  4 || i ==  8 || i == 23 ) printf(" " );
                printf (" %1x",pins[i] & 0xF ); };
}
copin(p) uchar *p;
{       int i; for (i=0;i<28;i++) { pins[i] = p[i]; };  }

ldsr(p) char *p;        /* Load pins into Hardware Shift Register         */
{int i; i=27; while ( i >= 0 ) { outportb(SCLK,p[i--] ); };
        outportb(STROBE,1) ;     /* Strobe all bits into BiMOS latches     */
        outportb(STROBE,0) ;
}
int bitinv(val,bits) int val,bits;       /* Invert n bits of val */
{       int res; res = 0;
        while (bits) { res = res + res + (val % 2); val = val / 2; bits--;
        };  return(res);
}
erra()
{}
/*
breaker()
{       bdos(9,"Break!!!\r\n$"); exit(0);
}
*/
subr()
{
/*      seg = 0xC000; off = offset = 0; byte = 0xaa; word = 0xAAAA;

           pokew(offset,seg,word);
        n = peek(offset,seg);
*/
        word = word; /*         offset = ++offset & 0x000F;      */
}
```

LISTING
Contributed by: Robert A. Freedman
"A PAL Programmer," by Robert A. Freedman. January, page 263.

```
/*      ZAPAL.C - Byte Magazine ZAP-A-PAL Programmer for IBM-PC            */


/*      Version 1.9 - (C) by Robert A. Freedman - 23 Oct 1986 - 8:00 PM */

#define base  0x100
#define DAC_A base+0
#define DAC_B base+1
#define SCLK  base+2

#define STROBE base+0x8
#define ENAB   base+0x9          /* Enable BiMOS drivers */
#define ENCH   base+0xA
#define ENCL   base+0xB
#define VLH    base+0xD
#define VINHIB base+0xE
#define TRIG   base+0xF

#define BUSY             (inportb(base+0xC) & 1)

static int verpin,vad,fuse;      /* Pin # to verify, I/O adr, State        */
static int veradr[10] = {9,7,6,5,4,3,2,1,0,8};  /* Mux adr for Pins 14 - 23 */

uchar pins[32]; /* Set up pin values here, then shift out to hardware   */

/*              Outputs         Control                Inputs
         2 2 2 1 1 1 1 1 2 1    1                             1 1
         2 1 0 9 8 7 6 5 3 4  1 3             2 3 4 5 6 7 8 9 0 1      */
static uchar clear[28] =
        {0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0}; /* Clear
*/
```

continued

```
static uchar odlo[28] =
        {0,0,0,0,0,0,0,0,0,0, 2,1,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0}; /* OD lo
*/
static uchar odhi[28] =
        {0,0,0,0,0,0,0,0,0,0, 1,2,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0}; /* OD hi
*/

static int pind[24] =   /* Maps Pin numbers to Shift Register Position  */
        {10,18,19,20,21,22,23,24,25,26,27,28,11,9,7,6,5,4,3,2,1,0,8};

int     n,i,lr,ix,ino,af,T20,fuzno;

int do_a_fuz(fuzno) int fuzno;  /* Set up to read or write a fuze */
{       int half,pin;
        pin = ( fuzno / ( T20 ? 32 : 40 ) );    /* Product-Line #  */

        outportb(ENAB,0);       /* Disable BiMOS drivers        */
        outportb(ENCL,1);       /* Disable BiMOS drivers CLOCK  */
        outportb(ENCH,1);       /* Disable BiMOS drivers OD     */

        half = ( T20 ? 32 : 40 );       /* Set OD and CLOCK pins        */
        pin( 1, (pin >= half?1:2) );    pin(13, (pin >= half?2:1) );

        (pin >= half ? ldsr(odhi) : ldsr(odlo) ); /* Shift OD & Clock */

        outportb(ENCL,0);       /* Enable BiMOS drivers CLOCK   */
        outportb(ENCH,0);       /* Enable BiMOS drivers OD      */

        selfuz(fuzno);  ldsr(pins);     /* Set up and load Shift-registers */

        outportb(ENAB,1);       /* Enable BiMOS drivers */

        return( verifuz() );    /* Read and return state of addressed fuze */
}
zot()   /* TRIGger the timing PAL to zap the fuze       */
{               while (  BUSY ) { outportb(TRIG,0); };
                while ( !BUSY ) { outportb(TRIG,1); };
                while (  BUSY ) { outportb(TRIG,0); };
                return(0);
}
int verifuz()   /*      Return state of fuze    */
{
        /* Assume the shift-registers are all set up by selfuz(fuzno);  */
        outportb(ENCL,1);       /* Pulse CLOCK pin by floating  */
        outportb(ENCL,0);       /* CLOCK to Z momentarily       */

        vad = veradr[verpin-14] + base; /* Compute Mux adr of Pin       */
        fuse = inportb(vad) & 1;        /* Read the state of the fuse   */
        /* On 16L8, 16R8 etc PALs, 0 = Blown, 1 = Intact fuse   */

        return(fuse);
}
selfuz(fuzno) int fuzno; /* Analyzes fuze-number and sets up all pins   */
{       int an,half,of,ox,lino,pl,pin,i;

        half = ( T20 ? 32 : 40 ); /* T20 is true for 20, false for 24 pin PAL
*/

        ldsr(clear);    /* Clear out old fuze info      */

        /*      Compute and place input pins    */

        lino =( fuzno % half );

        pin = ( fuzno / half );
        if (pin > (T20?63:79) ) return(ERROR);

        lr = 0; if   (lino & 2)  lr = 2;        /* Find which half      */

        ix = 0; if (!(lino & 1)) ix = 1;        /* Find the state of Pin x */

        /* Now find where to put the selected input pin, ie [I0..I9]    */

        for (i=0;i<10;i++) pin(2+i,2);  /* Pull all input pins to VIHH  */

        ino = lino / 4; pin(2+ino,ix);  /* Then set Selected pin to TTL */
        /*      Compute and Place Output Pins   */

        pin( 1, (pln >= half?1:2) );    /*      Set OD and CLOCK pins   */
        pin(13, (pln >= half?2:1) );

        pl = pln; if(pln >= half) pl = pln-half;
        an =  pl % 8;                           /* A0..An = pl mod 8    */
        ox = (pl / 8) & (T20?0xF:0x1F);         /* Select Outp Pin to pulse */
        for (i=14;i<=23;i++) { pin(i,0); };     /* Clear all Outputs    */

                             af = (T20?16:15 ); of = (T20?22:23 );
        if ( pln >= half ) { af = (T20?19:19 ); of = (T20?18:18 ); };
        if ((pln <  half) && !T20 ) an = bitinv(an,4);
        an = an & (T20? 7 : 0xF );

        for ( i = (T20?2:3); i >= 0; i--) {  /* Set Address bits  */
                pin(af+i,( an % 2 ? 2 : 0 ) ); an = an / 2;     };

        pin(of-ox,4);           /* Set Output Pin to Pulse              */
        verpin = of-ox;         /* Save pin to verify fuse state        */
        pin( (pln < half ? (T20?15:14) : (T20?22:23) ),lr); /* Set L/R  */
}       /* Now all the pins are set for programming or verification     */

int pin(n,val) int n,val;       /* Read or Store value of a pin */
{int v; uchar *p;       if (n == 0 || n > 24 ) val = 0xE;
                        p = pins + *(pind + n - 1 );
                        v = *p; *p = val; return(v);
}
ldsr(p) char *p;        /* Load pins into Hardware Shift Register       */
{       int i; i=27; while ( i >= 0 ) { outportb(SCLK,p[i--] ); };
        outportb(STROBE,1) ;    /* Strobe all bits into BiMOS latches   */
        outportb(STROBE,0) ;
}
```

PROFILER.ASM
Contributed by: Byron Sheppard
"High Performance Software Analysis on the IBM PC," by Byron Sheppard. January, page 157.

```
;
;TITLE: Profiler_timer
;
;DESCRIPTION: Fully compensated, high resolution
;timer.
;Internal timing resolution = 838ns.
;
;AUTHOR: Byron Sheppard
;
;CALLING SEQUENCE: call TIMER_START (FAR call)
;                  code to be timed
;                  call TIMER_STOP  (FAR call)
;
;INPUT: none
;
;OUTPUT: Display of elapsed time between the
;TIMER_START call and the TIMER_STOP call.
;
;REGISTERS CRASHED: none
;
;STACK REQUIREMENTS: 10 bytes
;
;CONDITION OF INTERRUPTS: TIMER_START = no change
;  TIMER_STOP  = variable, exit on
;
;EXTERNAL REFERENCES:
;       Procedures:     none
;       Data:   none
;
;CONFIDENCE QUOTIENT FOR:
;       Debugged:       average
;       Speed:  n/a
;       Elegance:       average
;
```

continued

```
;SPECIAL NOTES: -Counter 0 is used and must not
;be modified in the interval between the two
;timer calls.
; -All DOS timekeeping functions will operate
;as normal.
; -Timing events > 54.925 milli-sec requires
;interrupts ON.
; -PROFILER_TIMER does not affect code under test.
;
; Data segment = word combinable as DATASEG
; Code segment = byte combinable as CODESEG
;
; PROFILER_TIMER starts here
;-----------------------------------------------
DATASEG SEGMENT WORD PUBLIC
timer_low       equ ds:[006ch]
bios_dataseg    equ     0040h
timer_mode      equ     43h
timer0  equ     40h
count           dw      0       ;no. of interrupt ticks
                                ; (54.925millsec)
count_micro     dw      0       ;calc. from interrupt ticks
count_milli     dw      0       ;calc. from interrupt ticks
timer_micro     dw      0       ;from 8253 countdown...
                                ;...also final value
timer_milli     dw      0       ;final value
timer_sec       dw      0       ;final value
max_count       dw      65535   ;65536 ticks in a full count
adjustm         dw      67      ;compensation factor
timer_convert   dw      8381    ;838.096 nsec per tick
count_convert   dw      54925   ;54.925 milli-sec per count
ten_thousand    dw      10000
five_thousand   dw      5000
thousand        dw      1000
ten     dw      10
message_sec     db      'Seconds: ','$'
message_milli   db      'Milli-seconds: ','$'
message_micro   db      'Micro-seconds: ','$'
ascii_string    db   5 dup('d'),0dh,0ah,'$'
DATASEG ENDS



;--------------
; print macro
;--------------
print_string    macro   ;DOS function call to print string
        mov     ah,9    ;  pointed to by DS:DX
        int     21h
        endm

        public timer_start, timer_stop, bin_asc
CODESEG SEGMENT BYTE PUBLIC
        assume cs:codeseg, ds:dataseg

;-------------------------
;   timer_start routine
;   No parameters required
;-------------------------
timer_start     proc far
        push ax
        push dx
        push    ds

        mov     dx,dataseg ;point to my own data segment
        mov     ds,dx
        mov     timer_micro,0
        mov     timer_milli,0
        mov     timer_sec,0

;---- initialize counter 0 of 8253 timer  ---
        mov al,00110100B        ;ctr 0, LSB then MSB,
                                ; mode 2, binary
        out timer_mode,al       ;mode register for 8253
        sub ax,ax       ;0 results in max count
        out timer0,al   ;LSB first
        out timer0,al   ;MSB next

;--- read current BIOS time-of-day ---
        mov     dx,bios_dataseg ;point to BIOS data segment
        mov     ds,dx
        mov     ax,timer_low    ;get count
        mov     dx,dataseg      ;point to my own
                                ; data segment
        mov     ds,dx
        mov     count,ax        ;save count

        pop     ds
        pop     dx
        pop     ax
        ret
timer_start     endp


;---------------------------
;    TIMER_STOP routine
;    no parameters required
;---------------------------
timer_stop      proc    far
        push ax
        push    bx
        push dx
        push    ds      ;save user's DS

        mov     ax,dataseg      ;point to my own
        mov     ds,ax

;----------------------------------
; elapsed time since TIMER_START
; consists of:
;   1) timer count intervals - 840ns
;   2) interrupt ticks - 54 ms
;----------------------------------
;---- read counter 0 of 8253 timer ----
        mov     al,00h  ;latch counter for read
        cli             ;interrupts off until
                        ; BIOS tod is read
        out     timer_mode,al   ;8253 mode register
        in      al,timer0
        mov     dl,al
        in      al,timer0
        mov     dh,al   ;dx has 16 bit timer count

;--- calc the time due to 8253 counting ---
        mov     ax,max_count
        sub     ax,dx   ;timer count value
        mul     timer_convert   ;get in usable form
        div     ten_thousand    ;gives time in usec
        mov     timer_micro,ax  ;save usec, round nsec
        cmp     dx,five_thousand
        jb      cont    ;round down
        inc     timer_micro     ;round up

;--- get BIOS time due to interrupt ticks ----
cont:   mov     dx,bios_dataseg ;point to BIOS data segment
        mov     ds,dx
        mov     ax,timer_low
        mov     dx,dataseg      ;point to my own
                                ; data segment
        mov     ds,dx
        sti             ;interrupts ok now
        sub     ax,count        ;now have number of
                                ; 54 ms ticks
        mul     count_convert   ;get into usable form
        div  thousand
        mov     count_milli,ax  ;save milli sec part
        mov     count_micro,dx  ;save micro sec part

;---- check for jitter ----
        cmp     ax,0    ;check if elapsed time is "small"
        jne     jitter_ok       ;if not, then don't worry
                                ; about jitter
```

continued

```
         mov      ax,adjustm
         cmp      timer_micro,ax  ;
         jae      jitter_ok       ;if no jitter then ok
         mov      timer_micro,ax  ; else "-ve time artifact"
                                  ;  so fix

;---------------------------------
; combine the timer and count values
; put result in timer variables
;---------------------------------
jitter_ok:        mov    ax,dx    ;get count_micro
         add      ax,timer_micro  ;sum micro fields
         cmp      ax,adjustm      ;check for underflow
                                  ; possibility
         jae      compensate      ;  go ahead - safe
         dec      count_milli     ;  borrow
         add      ax,1000
compensate:
         sub      ax,adjustm ;compensate for time delays
         mov      timer_micro,ax
         cmp      ax,1000 ;check for field overflow
         jb       fld_ok  ;timer_micro field ok
         sub      dx,dx   ;timer_micro too large
         div      thousand        ;so carry out
                                  ; into timer_milli
         mov      timer_milli,ax
         mov      timer_micro,dx
fld_ok:  mov      ax,count_milli  ;sum milli fields
         add      timer_milli,ax
         cmp      timer_milli,1000        ;check as above
         jb       display
         sub      dx,dx
         mov      ax,timer_milli
         div      thousand
         mov      timer_sec,ax
         mov      timer_milli,dx


;------ Display results -----
display:
         lea      dx,message_sec  ;display seconds header
         print_string
         lea      bx,ascii_string ;convert seconds in ascii
         mov      ax,timer_sec
         call     bin_asc
         mov      dx,bx   ;bx points to converted ascii string
         print_string     ;display seconds

         lea      dx,message_milli ;display milli-seconds
                                   ; header
         print_string
         lea      bx,ascii_string ;convert milli-seconds
                                  ; in ascii
         mov      ax,timer_milli
         call     bin_asc
         mov      dx,bx
         print_string     ;display milli-seconds

         lea      dx,message_micro ;display micro-seconds
                                   ; header
         print_string
         lea      bx,ascii_string ;convert micro-seconds
                                  ; in ascii
         mov      ax,timer_micro
         call     bin_asc
         mov      dx,bx
         print_string     ;display micro-seconds

         pop      ds      ;restore user's DS
         pop      dx
         pop      bx
         pop      ax
         ret
timer_stop        endp
```

```
;----------------------------------
; Binary to Ascii conversion routine
;   Successive division by 10
;   Store remainder
; Entry:
;   BX = pointer to string buffer
;   AX = unsigned binary number
; Exit:
;   BX = ptr to Ascii number
;----------------------------------
bin_asc proc near
        push    dx
        push cx
        push ax

        mov     cx,5    ;clear string buffer
clear_buf:      mov     byte ptr [bx],30h
        inc     bx
        loop    clear_buf

convert:        sub     dx,dx   ;clear upper half
                                ; of dividend
        div     ten     ;(dx:ax)/10
        add     dx,30h  ;convert decimal digit
                        ; to ascii digit
        dec     bx
        mov     [bx],dl ;save character
        or      ax,ax
        jnz     convert ;finished?

        pop     ax
        pop     cx
        pop     dx
        ret
bin_asc endp

CODESEG ENDS
        end
```

RGNMAKER.ASM
Contributed by: Howard Katz
"Region Maker," by Howard Katz. January, page 145.

```
;  RgnMaker.ASM         Sun 23 Feb '86  h. katz
;                       Tue 22 July


XREF DoContour                          ; source in < Traverse.ASM >
XREF Save_To_File, PutFile_Posit        ; source in < SaveRgn.ASM >
XREF Have_Prior_DITLs, ItemHit

XDEF GetFirstPixel, FormingRgn, StartCoords, Trav_Count, MyRowBytes
XDEF IsRegion, CreateMenu, WMgrPort, ScratchSTR, PenPoint, WStorage
XDEF Save_Image, Restore_Image, Stop_Alert, RgnHandle, UnHiLite

INCLUDE QuickEqu.D
        ; portRect      equu    16      ; offset in Window Record
        ; Bounds        equ     6       ; offsets into BitMap
        ; rowBytes      equ     4
        ; Top           equ     0       ; offsets into portRect & Bounds
        ; Left          equ     2
        ; Bottom        equ     4
        ; Right         equ     6

StringToNum     equ     1       ; selector for _Pack7

scratch8        equ     $9FA
scratch20       equ     $1E4
ScreenBase      equ     $824

ScrapSize       equ     $960    ; ( word ) size in bytes
ScrapHandle     equ     $964
ScrapCount      equ     $968    ; ( word ) current counter value
ScrapState      equ     $96A    ; ( word ) + = on disk
                                ;  0 = in mem / - = not inited
```

continued

```
ScrapName        equ     $96C     ;  StringPtr

CmdKey           equ     8        ; BitNum in Event Modifier Record
PutFile_ID       equ     -3999    ; ResID for SFPutFile Dialog

WindID           equ     100

AppleMenu        equ     1

FileRgnMenu      equ     2
  WriteRgn_Item  equ     1
  Quit_Item      equ     3

EditRgnMenu      equ     3
  CutItem        equ     3
  CopyItem       equ     4
  PasteItem      equ     5

CreateRgnMenu            equ     4
  BuildRgnItem           equ     1
  DontBuildItem          equ     2
  CopyRegionItem         equ     4
  AddItem                equ     5
  SubtractItem           equ     6

DisplayRgnMenu   equ     5
LastMenu         equ     5

FirstDACNum      equ     3

HiBitNum         equ     7        ; working with bytes

MouseDownEvent   equ     1
KeyDownEvent     equ     3

Traverse_Cursor  equ     10       ; ResNum of my cursor


INCLUDE MacTraps.D

        st       FormingRgn(a5)          ; turn off if we don't want to
                                         ; collect _Line's into a RgnDef
        sf       DeskAcc_Opened(a5)
        sf       Have_Pasted(a5)
        st       First_Activate(a5)
        sf       DoneFlag(a5)
        sf       IsRegion(a5)            ; We've not done a Traverse with
                                         ;  'Form Region' On
        sf       DoneCopyRegion(a5)      ; We've not copied a Region Traverse
                                         ;   to the Work Area

        sf       Have_Prior_DITLs(a5)    ; no prior saving of DITLs
        move.l   #0, BitMap(a5)          ; haven't done a CopyBits yet

        BSR      InitManagers
        BSR      Save_WMgrPort
        BSR      Install_Menus
        BSR      OpenWindow

EventLoop

        _SystemTask

        tst.b    Have_Pasted(a5)         ; if we haven't Pasted,
        beq      @1                      ; leave the cursor alone

        clr.l    -(sp)
        _FrontWindow                     ; which window is frontmost ?
        move.l   (sp)+, a0
        lea      WStorage, a2            ; get Ptr to the Contour Window
        cmpa.l   a0, a2
        beq.s    @Check_pRect            ; Contour Window is in front

        bra.s    @1                      ; some other window is in front
```

```
@Check_pRect

        clr     -(sp)              ; space for Boolean Func Result
        pea     MouseLoc
        _GetMouse                  ; where's the cursor ?
        move.l  MouseLoc, -(sp)
        pea     portRect(a2)
        _PtInRect
        tst     (sp)+
        beq.s   @1                 ; NOT in contentRgn of Contour Window

        bsr     Set_Traverse_Cursor
        bra.s   @2

@1      _InitCursor

@2      bsr     GetNextEvent
        tst.b   DoneFlag(a5)
        beq.s   EventLoop
        _ExitToShell


Set_Traverse_Cursor

        clr.l   -(sp)                      ; reserve space for Handle
        move    #Traverse_Cursor, -(sp)    ; Cursor ID
        _GetCursor
        move.l  (sp)+, a0
        move.l  (a0), a0                   ; ptr to CursorData
        move.l  a0, -(sp)
        _SetCursor

        RTS


GetNextEvent

        subq.l  #2, sp
        move    #-1, -(sp)                 ; eventMask = everyEvent
        pea     EventRecord
        _GetNextEvent
        tst.b   (sp)+
        beq.s   Return
        move    What, D0
        beq.s   Return
        cmp     #9, d0                     ; don't worry about events
        bge.s   Return                     ; numbered 9 or higher

        add     d0, d0
        lea     EventJTable, a0
        add     0(a0, d0), a0
        jmp(a0)                            ; we RTS out of each routine
                                           ; to EventLoop

EventJTable

@NullEvent      dc      Return          - EventJTable   ; 0
@MDown          dc      MouseDown       - EventJTable   ; 1
@MouseUp        dc      Return          - EventJTable   ; 2
@KDown          dc      KeyDown         - EventJTable   ; 3
@KeyUp          dc      Return          - EventJTable   ; 4
@AutoKey        dc      Return          - EventJTable   ; 5
@Update         dc      Return          - EventJTable   ; 6
@Disk           dc      Return          - EventJTable   ; 7
@Activate       dc      Activate        - EventJTable   ; 8
@Undefined      dc      Return          - EventJTable   ; 9

Activate

        ; check if the Contour Window is coming active.  If it is,
        ; and this is the first time here, try Pasting in from the Scrap.
        ; else check if a Desk Accessory was open just prior.  If it was,
        ; restore the saved background
```

continued

```
        clr.l   -(sp)
        _FrontWindow                    ; which window is frontmost ?
        move.l  (sp)+, a0
        lea     WStorage, a2            ; get Ptr to the Contour Window
        cmpa.l  a0, a2
        bne.s   @Rts                    ; Contour Window

@0      tst.b   First_Activate(a5)      ; do an 'auto-Paste' on 1st Activate
        beq.s   @1                      ; no
        sf      First_Activate(a5)      ; yes
        BRA     Paste_From_Clip

@1      tst.b   DeskAcc_Opened(a5)      ; was User using the ScrapBook ?
        beq.s   @Rts                    ; no

        sf      DeskAcc_Opened(a5)      ; yes - turn off flag

        BSR     Restore_Image           ; restore old background

@Rts    RTS


MouseDown

        clr     -(sp)           ; returns where Mouse was clicked
        move.l  Where, -(sp)    ; global coords of Mouse Location
        pea     WhichWindow     ; whose ?
        _FindWindow
        move    (sp)+, d0       ; where was it
        add     d0, d0
        lea     MouseJTable, a0
        add     0(a0, d0), a0
        jmp(a0)

Return

        RTS                             ; return to EventLoop

MouseJTable

@inDesk         dc      Return          - MouseJTable
@inMBar         dc      InMenu          - MouseJTable
@SysEvent       dc      SystemEvent     - MouseJTable
@Content        dc      InContent       - MouseJTable
@Drag           dc      Return          - MouseJTable
@Grow           dc      Return          - MouseJTable
@GoAway         dc      TrackGoAway     - MouseJTable


Save_Image

        ; first get the intersection of the current port and the
        ; incoming Dialog or Alert in Global Coords.
        ; ResID of Resource in D3 / ResType in A3

        lea     WStorage+portRect, a0
        lea     Scratch20, a2
        move.l  (a0)+, (a2)+            ; move Window pRect into 1st 8
        move.l  (a0), (a2)+             ; bytes of Scratch20 area
        pea     Scratch20
        _LocalToGlobal
        pea     Scratch20+4
        _LocalToGlobal

        clr.l   -(sp)                   ; returned Handle
        move.l  a3, -(sp)               ; 'DLOG' or 'ALRT'
        move    d3, -(sp)               ; DLOG or ALRT ResID
        _GetResource
        move.l  (sp)+, d1               ; Handle
        BEQ     @ErrRts

@1      move.l  d1, a0
        move.l  (a0), a0                ; Ptr to Data

        move.l  (a0)+, (a2)+            ; move pRect of DLOG into 2nd 8 bytes
        move.l  (a0), (a2)              ; of Scratch20 area
        cmp     #PutFile_ID, d3
        bne.s   @2

        ; SFPutFile DLOG requires an adjustment for PutFile_Posit

        pea     Scratch20+8
        move.l  PutFile_Posit, -(sp)    ; Global TopLeft of DLOG pRect
        _OffsetRect

@2      pea     Scratch20+8             ; compensate for Window Frame
        move    #-8, -(sp)              ; around the DLOG/ALRT pRect
        move    #-8, -(sp)
        _InsetRect

        clr     -(sp)                   ; BOOLEAN result
        pea     Scratch20               ; Contour Window ( Global )
        pea     Scratch20+8             ; Dialog ( Global )
        pea     Scratch20+8             ; => Intersection ( Global )
        _SectRect
        tst     (sp)+
        BEQ     @ErrRts

        pea     Scratch20+8
        _GlobalToLocal
        pea     Scratch20+8+4
        _GlobalToLocal

        lea     BitMap(a5), a3
        move.l  Scratch20+8,  Bounds(a3)
        move.l  Scratch20+12, Bounds+4(a3)

        BSR     Get_Size                ; returns D0 = Size ( bytes )
                                        ;         D1 = rowBytes
        move    d1, rowBytes(a3)

        _NewHandle,CLEAR
        BMI     @ErrRts

        move.l  a0, Image_Handle(a5)
        move.l  (a0), (a3)              ; deref to Ptr = BaseAddr
                                        ;  in ToBitMap
        move.l  (a5), a2                ; QDVars
        move.l  (a2), a2
        lea     2(a2), a2

        move.l  a2, -(sp)               ; fromBitMap
        move.l  a3, -(sp)               ; toBitMap

        pea     Scratch20+8             ; local coords of fromRect
        move.l  (sp), -(sp)             ; again for ToRect
        move    #0, -(sp)               ; mode
        clr.l   -(sp)                   ; maskRgn = NIL
        _CopyBits
        BRA     @Rts


@ErrRts move.l  #0, BitMap(a5)          ; can't save the image - forget it

@Rts    RTS


Restore_Image

        pea     WStorage
        _BeginUpdate

        tst.l   BitMap(a5)              ; have we done a 'Save_Image' CopyBits
?
        beq.s   @End                    ; no

        lea     BitMap(a5), a2
        move.l  Image_Handle(a5), a0
        move.l  (a0), (a2)              ; baseAddr

        move.l  a2, -(sp)               ; fromBitMap
```

continued

```
        move.l  (a5), a3
        move.l  (a3), a3
        pea     2(a3)                   ; toBitMap ( the screen )

        pea     Bounds(a2)
        move.l  (sp), -(sp)             ; again for ToRect
        move    #0, -(sp)               ; mode
        clr.l   -(sp)                   ; maskRgn = NIL
        _CopyBits

        move.l  #0, BitMap(a5)          ; clear CopyBits 'flag'
        move.l  Image_Handle(a5), a0
        _DisposHandle                   ; clear up space on the Heap

@End    pea     WStorage
        _EndUpdate

        RTS


Get_Size

        move    Bounds+Right(a3), d2
        ext.l   d2
        sub     Bounds+Left(a3), d2     ; width
        add     #1, d2
        divu    #16, d2                 ; => remainder + quotient
        move    d2, d1                  ; save quotient
        and.l   #$FFFF0000, d2          ; check remainder
        beq.s   @1                      ; no remainder
        add     #1, d1                  ; rowWords

@1      add     d1, d1                  ; words -> bytes
        move    Bounds+Bottom(a3), d0
        sub     Bounds+Top(a3), d0
        add     #1, d0
                                        ; d1 => rowBytes
        mulu    d1, d0                  ; d0 => Size ( long ) for _NewHandle

        RTS


TrackGoAway

        clr     -(sp)                   ; space for BOOLEAN result
        pea     WStorage
        move.l  Where, -(sp)            ; from Event Record
        _TrackGoAway
        move    (sp)+, d0
        bne     SetDone                 ; mouse WAS released in goAway box
        RTS                             ; it wasn't

InMenu

        clr.l   -(sp)
        move.l  Where, -(sp)            ; global again
        _MenuSelect                     ; track the mouse in the MBar
        move.l  (sp)+, d0               ; save Menu and ItemNum
        swap    d0                      ; MenuNum -> LowByte

FindMenu

        cmp     #AppleMenu, d0
        BEQ     In_Apple_Menu
        cmp     #FileRgnMenu, d0
        BEQ     In_File_Menu
        cmp     #EditRgnMenu, d0
        BEQ     In_Edit_Menu
        cmp     #CreateRgnMenu, d0
        BEQ     In_CreateRgn_Menu
        cmp     #DisplayRgnMenu, d0
        BEQ     In_DisplayRgn_Menu
        BRA     UnHiLite

In_Apple_Menu
```

```
        swap    d0              ; ItemNum back in Low Byte
        cmp     #1, d0
        bne     @GetDAC         ; not Item 1 - must be Open a Desk Acc

        move    #107, d3        ; ResID of upcoming 'About' DLOG
        move.l  #'DLOG', a3
        BSR     Save_Image

        clr.l   -(sp)           ; reserve space for Ptr
        move    #106, -(sp)     ; 'About' DLOG ID
        move.l  #0, -(sp)       ; create space
        move.l  #-1, -(sp)      ; in front of everything
        _GetNewDialog
        move.l  (sp)+, d4

        BSR     @Wait_for_next

        move.l  d4, -(sp)
        _CloseDialog

        clr.l   -(sp)           ; reserve space for Ptr
        move    #107, -(sp)     ; 'About' DLOG Instructions ID
        move.l  #0, -(sp)       ; create space
        move.l  #-1, -(sp)      ; in front of everything
        _GetNewDialog
        move.l  (sp)+, d4

        BSR     @Wait_for_next

        move.l  d4, -(sp)
        _CloseDialog

        clr.l   -(sp)           ; reserve space for Ptr
        move    #108, -(sp)     ; 'About' DLOG Instructions Part2 ID
        move.l  #0, -(sp)       ; create space
        move.l  #-1, -(sp)      ; in front of everything
        _GetNewDialog
        move.l  (sp)+, d4

        BSR     @Wait_for_next

        move.l  d4, -(sp)
        _CloseDialog

        BSR     Restore_Image

        bra.s   UnHiLite

@Wait_for_next

        move.l  #0, -(sp)       ; no filterProc
        pea     ItemHit
        _ModalDialog

        move    ItemHit, d0
        cmp     #1, d0
        bne     @Wait_for_next

        RTS

@GetDAC

        move.l  HAppleMenu(a5), -(sp)   ; saved menuHandle for AppleMenu
        move    d0, -(sp)
        pea     DACName
        _GetItem

        st      DeskAcc_Opened(a5)

        move    #WindID, d3             ; ResID of upcoming Dialog/Alert
        move.l  #'WIND', a3
        BSR     Save_Image              ; save Bits to be hidden by the
DeskAcc

        clr     -(sp)
```

continued

```
        pea     DACName
        _OpenDeskAcc
        move    (sp)+, d0

        pea     WStorage
        _SetPort

UnHiLite
        clr     -(sp)
        _HiLiteMenu
        RTS


In_File_Menu

        swap    D0
        cmp     #Quit_Item, D0
        BEQ     SetDone
        cmp     #WriteRgn_Item, d0
        bne     UnHiLite

        BSR     Save_To_File    ; see source in < SaveRgn.ASM >

        bra     UnHiLite


In_Edit_Menu

        swap    d0                      ; Put ItemNum in Low Byte
        move    d0, d3
        BSR     System_Edit
        BNE.s   @Bra                    ; Desk Acc handled the Menu selection
        move    d3, d0                  ; restore MenuID & ItemNum

        cmp     #CutItem, d0
        BEQ.s   @Cut
        cmp     #CopyItem, d0
        BEQ.s   @Copy
        cmp     #PasteItem, d0
        BNE.s   @Bra

; do a Paste

        BRA     Paste_From_Clip

@Cut

        BRA     @Bra

@Copy

@Bra    BRA     UnHiLite


System_Edit

        sub     #1, d0                  ; check if the Desk Accessory is
        clr     -(sp)                   ; going to handle our Edit selection
        move    d0, -(sp)
        _SysEdit
        move    (sp)+, d0               ; pop the result
        RTS                             ; ( FALSE = WE handle it )


In_DisplayRgn_Menu

        swap    d0
        cmp     #1, d0                  ; Clear Window ?
        beq.s   @clear

        tst.b   DoneCopyRegion(a5)      ; Don't Allow Region Operations if
        BEQ     @bra1                   ;  we haven't copied one to Work Area

        cmp     #3, d0                  ; Frame Region ?
        beq.s   @Frame
        cmp     #4, d0                  ; Paint Region ?
        beq.s   @Paint
        cmp     #5, d0                  ; Invert Region ?
        beq.s   @Invert
        cmp     #7, d0
        beq.s   @DrawSize
        BRA     @Bra1

@Frame  move.l  RgnHandle(a5), -(sp)
        _FrameRgn
        bra.s   @Bra1

@Paint  move.l  RgnHandle(a5), -(sp)
        _PaintRgn
        bra.s   @Bra1

@Invert move.l  RgnHandle(a5), -(sp)
        _InverRgn
        bra.s   @Bra1

@clear  move.l  (a5), a0
        move.l  (a0), a0
        pea     16(a0)
        _EraseRect
        bra.s   @Bra1

@DrawSize

        move.l  WMgrPort, -(sp)
        _SetPort
        move.l  PenPoint, -(sp)
        _MoveTo
        pea     '
        _DrawString

        move.l  RgnHandle(a5), a0       ; Handle
        move.l  (a0), a0                ; Pointer to ( Addr of ) Region
        move    (a0), d0                ; RgnSize ( INT )

        ext.l   d0
        lea     ScratchSTR, a0
        move    #0, -(sp)               ; Num to String
        _Pack7
        pea     ScratchSTR
        _DrawString
        pea     ' Bytes'
        _DrawString

        pea     WStorage
        _SetPort


@bra1   BRA     UnHiLite


SetDone
        st      DoneFlag(a5)
        RTS


In_CreateRgn_Menu

        swap    D0
        move    D0, D3                  ; Save Selected Item Number

        cmp     #DontBuildItem, d0
        bhi     @2                      ; Was it 'Copy' or greater ?

; User Selected either 'Build Region' or 'Display Only'

        move.l  CreateMenu(A5), -(sp)   ; UnCheck Both Items 1 & 2
        move    #1, -(sp)
        sf      -(sp)
        _CheckItem
        move.l  CreateMenu(A5), -(sp)
        move    #2, -(sp)
```

continued

```
        sf      -(sp)
        _CheckItem

        move.l  CreateMenu(A5), -(sp)   ; and Check the Apt Item
        move    d3, -(sp)
        st      -(sp)
        _CheckItem

        cmp     #DontBuildItem, d3
        bne.s   @1                      ; We're ARE Forming a Region
        sf      FormingRgn(A5)          ; We're NOT Forming a Region
        BRA     @Bra

@1      st      FormingRgn(A5)
        BRA     @Bra

; User Selected either 'Copy to', 'Add to', or 'Subtract from Work Area'

@2      tst.b   IsRegion(a5)
        BEQ     @Bra                    ; We haven't Formed a Region yet

        cmp     #CopyRegionItem, d3
        beq.s   @CopyToWorkArea

        tst.b   DoneCopyRegion(a5)
        BEQ     @Bra                    ; We can't Add or Subtract from the
                                        ;  Work Area until we've done a 'Copy'
        cmp     #AddItem, d3
        beq.s   @AddToWorkArea
        cmp     #SubtractItem, d3
        beq.s   @SubtractFromWorkArea

        BRA     @Bra

@CopyToWorkArea

        tst.b   DoneCopyRegion(a5)      ; have we previously done a 'Copy' ?
        beq.s   @3                      ; no

        move.l  RgnHandle(a5), -(sp)    ; we don't want old Regions cluttering
        _DisposRgn                      ; up the Heap

@3      move.l  a6, -(sp)               ; Handle to Current (traverse) Region
        clr.l   -(sp)
        _NewRgn
        move.l  (sp), RgnHandle(a5)
        _CopyRgn

        st      DoneCopyRegion(a5)      ; use this flag to allow
                                        ; later Add & Subtract
        move.l  FileMenu(A5), -(sp)
        move    #WriteRgn_Item, -(sp)
        _EnableItem

        move.l  CreateMenu(A5), -(sp)
        move.l  (sp), -(sp)
        move    #AddItem, -(sp)         ; And allow these 2 selections
        _EnableItem
        move    #SubtractItem, -(sp)
        _EnableItem

        move.l  DisplayMenu(A5), -(sp)
        move.l  (sp), -(sp)
        move.l  (sp), -(sp)
        move.l  (sp), -(sp)
        move    #3, -(sp)               ; Frame Region
        _EnableItem
        move    #4, -(sp)               ; Paint Region
        _EnableItem
        move    #5, -(sp)               ; Invert Region
        _EnableItem
        move    #7, -(sp)               ; Draw Region Size
        _EnableItem

        BRA     @Bra
```

```
@AddToWorkArea

        move.l  a6, -(sp)               ; SourceA = Current (traverse) Region
        move.l  RgnHandle(a5), -(sp)    ; SourceB = Destination (Work Area)
Rgn
        move.l  RgnHandle(a5), -(sp)    ; Destination = Work Area
        _UnionRgn

        BRA.s   @Bra

@SubtractFromWorkArea

        move.l  RgnHandle(a5), -(sp)    ;   Region B
        move.l  a6, -(sp)               ; - Region A
        move.l  RgnHandle(a5), -(sp)
        _DiffRgn

@Bra    BRA     UnHiLite


SystemEvent

        pea     EventRecord
        move.l  WhichWindow, -(sp)
        _SystemClick
        RTS


InContent

        tst.b   Have_Pasted(a5)         ; have we got anything to traverse ?
        beq     Return                  ; no

        bra     DoContour               ; ( returns from DoContour to
EventLoop )


KeyDown
        ; check to see if the Command Key was down
        ; if so, see it it's a menu-item equivalent
        ; else ignore it

        move    Modifiers, d3
        btst    #CmdKey, d3
        bNE     @GetCmdKey

        RTS

@GetCmdKey

        clr.l   -(sp)
        move    Message+2, -(sp)        ; get the character
        _MenuKey
        move.l  (sp)+, d0
        swap    d0                      ; put MenuID in Low Byte

        BRA     FindMenu


Paste_From_Clip

        ; check the scrap.  if there's no PICT there, just
        ; beep and return for now.  else frame it in the window

        move.l  ScrapHandle, d0
        beq     @8                      ; Beep no Scrap

        ; we've got a 'PICT' on the Clipboard

        move.l  #0, d0
        _NewHandle
        bmi     ErrReturn
```

continued

```
        move.l  a0, a2                  ; save the handle
        clr.l   -(sp)
        move.l  a2, -(sp)               ; destination handle for incoming
        move.l  #'PICT', -(sp)          ; PICT
        pea     Offset
        _GetScrap
        tst.l   (sp)+                   ; byte count or OS ErrCode
        bmi     @8                      ; Beep No Data of Type in Scrap

        move.l  a2, -(sp)               ; Handle to PICTure ( used in 3rd call

        move.l  a2, a6                  ; save the Handle

        move.l  (a2), a2                ; Ptr to PICT Resource
        lea     2(a2), a2               ; Ptr to PicFrame
        lea     Scratch8, a3
        move.l  (a2)+, (a3)+            ; A4 =  PicFrame.TopLeft
        move.l  (a2), (a3)              ;               .BottomRight
        lea     Scratch20, a4
        lea     WStorage, a2
        move.l  portRect+Left(a2), d3   ; pRect.LeftBottom
        add.l   #$000AFFF6, d3          ; move PICT Org over ( 10, -10 )
        move.l  d3, (a4)
        move.l  -2(a3), -(sp)           ; PicFrame.Left
        pea     (a4)                    ; pRect.LeftBottom  = destPoint
        _SubPt

        move.l  (a4), d0
        swap    d0
        move.l  d0, (a4)

        pea     Scratch8
        move.l  (a4), -(sp)
        _OffsetRect

        pea     WStorage+portRect       ; erase any old PICTs
        _EraseRect

        pea     -4(a3)                  ; destRect = picFrame rectangle
        _DrawPicture

        move.l  a6, -(sp)               ; free up some space in the heap
        _KillPicture

        tst.b   Have_Pasted(a5)         ; reset to our Cursor if 1st time
        bne.s   @9                      ; not 1st time

        st      Have_Pasted(a5)
        bra     @9

@8      move    #2, -(sp)               ; Beep if couldn't Paste
        _SysBeep

@9      BRA     UnHiLite

ErrReturn
        _Debugger

Offset  dc.l    0                       ; used if we need most 'important'
                                        ; version of Type

GetFirstPixel

        ; user points w/mouse to a point just to the left of any
        ; left-edged pixel ( no 6-Neighbor ) in the region to be traversed.
        ; Program then scans left to right to find the apt byte and BitNum.

        ; < Where > is global coords of point to left of ON pixel in Rgn

        ; now calculate byte offset to first pixel in block from
        ; global address


        pea     Where
        _GlobalToLocal
        lea     WStorage, a1            ; set up Right Bounds to check
        move    portRect+Right(a1), d3  ; against ( else we erase the
        sub     #4, d3                  ; Window Frame )
                                        ; and leave a little leeway
@CheckNextPixel

        move.l  Where, -(sp)
        _MoveTo
        clr.b   -(sp)
        move.l  Where, -(sp)
        _GetPixel

        tst.b   (sp)+
        bne.s   @2                      ; we found one !
        lea     Where, a0
        add     #1, 2(a0)               ; move horizontally 1
        cmp     2(a0), d3               ; have we passed the right edge ?
        bpl.s   @CheckNextPixel         ; no

@1      _ShowCursor                     ; yes - we got problems

        move    #102, d3                ; StrID for upcoming Alert
        BSR     Stop_Alert              ; 'Can't locate first point'

        RTS


@2      move.l  Where, StartCoords(a5)
                                ; save the local Where for an _OpenRgn
                                ; _MoveTo command & for closing the loop

        pea     Where           ; now that we've found the local coords of
        _LocalToGlobal          ; our first pixel, convert that to an
        move.l  Where, d0       ; absolute memory reference ( Addr + BitNum )

        ; get byte addr ( a1 ) and bitNum of point ( d1 )

        swap    d0                      ; y in Low Word
        move    d0, d1
        mulu    #64, d1                 ; number of bytes down
        swap    d0                      ; x back in Low Word
        and.l   #$0000FFFF, d0          ; zero Hi Word
        divu    #8, d0                  ; number of bytes over
        add     d0, d1
        add.l   ScreenBase, d1
        move.l  d1, A3                  ; = < Addr of StartPt >

        swap    d0                      ; get back remainder in pixels
        and.l   #$0000FFFF, d0          ; zero quotient
        sub.b   #7, d0
        neg      d0
        and.l   #7, d0

        move    d0, D3                  ; = < BitNum of StartPt >
        move    #-1, d0                 ; Neg Flag = Found One

        RTS


Stop_Alert      ; an Alert of some sort is coming up
                ; the ID of the String DITL is in D3

        move.l  #'ALRT', a3
        BSR     Param_Text
        move    #101, d3        ; ResID for all StopAlerts
        BSR     Save_Image

        _InitCursor             ; reset to the standard northwest arrow

        clr     -(sp)
        move    d3, -(sp)       ; AlertID
        move.l  #0, -(sp)
        _StopAlert
        move    (sp)+, d0
```

continued

```
        BSR     Restore_Image

        RTS


Param_Text

        clr.l   -(sp)
        move    d3, -(sp)
        _GetString
        move.l  (sp)+, a0
        move.l  (a0), -(sp)      ; ^0
        move.l  #0, -(sp)        ; ^1
        move.l  #0, -(sp)        ; ^2
        move.l  #0, -(sp)        ; ^3
        _ParamText

        RTS


OpenWindow

        clr.l   -(sp)            ; for returned WindowPtr
        move    #WindID, -(sp)   ; WindowID
        pea     WStorage         ; storage for Window Record
        move.l  #-1, -(sp)       ; in front
        _GetNewWindow
        _SetPort
        lea     WStorage, a0
        pea     portRect(a0)     ; push the addr of portRect
        _ClipRect
        RTS

Save_WMgrPort

        move.l  (a5), a0
        move.l  (a0), a0         ; thePort
        lea     WMgrPort, a1
        move.l  a0, (a1)
        move    ScreenBits+rowBytes(a0), MyRowBytes(a5)
        RTS


Install_Menus

        clr.l   -(sp)
        move    #AppleMenu, -(sp)
        _GetRMenu                       ; resNum of DeskAcc Menu
        move.l  (sp), HAppleMenu(a5)    ; save MenuHandle for later
        move.l  (sp), -(sp)             ; push copy for _AddResMenu
        clr     -(sp)                   ; append to end
        _InsertMenu

        move.l  #'DRVR', -(sp)
        _AddResMenu

        clr.l   -(sp)                   ; Note:  would have been a bit more
        move    #FileRgnMenu, -(sp)     ; elegant to store these handles
        _GetRMenu                       ; in an array rather than separate
        move.l  (sp), FileMenu(a5)      ; variables.  Oh well, if it works ...
        clr     -(sp)
        _InsertMenu

        clr.l   -(sp)
        move    #EditRgnMenu, -(sp)
        _GetRMenu
        move.l  (sp), EditMenu(a5)
        clr     -(sp)
        _InsertMenu

        clr.l   -(sp)
        move    #CreateRgnMenu, -(sp)
        _GetRMenu
        move.l  (sp), CreateMenu(a5)
        clr     -(sp)
        _InsertMenu
        clr.l   -(sp)
        move    #DisplayRgnMenu, -(sp)
        _GetRMenu
        move.l  (sp), DisplayMenu(a5)
        clr     -(sp)
        _InsertMenu

        _DrawMenuBar

        move.l  CreateMenu(A5), -(sp)
        move    #BuildRgnItem, -(sp)    ; check it
        st      -(sp)
        _CheckItem

        RTS


InitManagers

        pea             -4(a5)
        _InitGraf
        _InitFonts
        _InitWindows
        _InitMenus
        clr.l   -(sp)
        _InitDialogs
        _TEInit
        _InitCursor
        move.l  #$FFFF0000, d0
        _FlushEvents
        RTS

;  CONSTANTS ( PC-rel addressing ) ----------------------------------

EventRecord
   What:        dc.w    0
   Message:     dc.l    0
   When:        dc.l    0
   Where:       dc.l    0
   Modifiers:   dc.w    0

PenPoint        dc.l    0               ; start of Rgn Size Display
MouseLoc        dc.l    0               ; for _PtInRect cursor check

ScratchSTR      dcb.b   10, 0

WStorage                dcb.b   156, 0

DACName         dcb.b   40, 0

WhichWindow     dc.l    0
WMgrPort        dc.l    0

boundsRect      dc.w    45, 10, 335, 500

StandardProcs   dcb.l   13, 0


;  VARs ( ref'd off A5 ) ------------------------------------------

Trav_Count      ds      1

Default_Vol     ds      1       ; VolRefNum of Default Volume
CurResFile      ds      1       ; RefNum of Current Res File

MyRowBytes      ds.l    1       ; used in Traverse.ASM / copy of rowBytes
RgnHandle       ds.l    1
StartCoords     ds.l    1
Image_Handle    ds.l    1       ; Handle to saved Bit Image

HAppleMenu      ds.l    1       ; handles for menus
FileMenu        ds.l    1
EditMenu        ds.l    1
```

continued

```
CreateMenu      ds.l    1
DisplayMenu     ds.l    1

BitMap          ds.b    14

DeskAcc_Opened  ds.b    1         ; User has opened the ScrapBook DeskAcc
First_Activate  ds.b    1         ; for Pasting from the Scrap on 1st Activate
Have_Pasted     ds.b    1         ; have done a Paste
DoneFlag        ds.b    1         ; GoAwayBox Click or File Menu 'Quit'
FormingRgn      ds.b    1         ; Build / Don't Build a Region during Traverse
IsRegion        ds.b    1         ; we've done a Traverse w/ 'Form Region' On
DoneCopyRegion  ds.b    1         ; we've done a 'Copy Region' to the Work Area

        END
```

RELXH.TXT
Contributed by: Gregg Williams
"An Introduction to Relaxation Methods," by Gregg Williams. January, page 111.

About the Programs

Listings 1 and 2 enable you to experiment with the relaxation method if you have a computer that runs Microsoft BASIC. These listings, as is, run on an Apple II computer in 40-column mode, but only the subroutines at lines 10000 and 14000 in listing 1 and lines 10000 and 23000 in listing 2 (which implement file input from and output to disk) must be changed to get this program to work on machines like the Radio Shack Model IV, the IBM Personal Computer, the Commodore 64 and VIC, and other computers.

You may want to change the print-array routines at line 20000 in each program to display data in the best way for your computer; these routines were written to display data neatly on an 80-column printout. All REM statements can be removed, and all variables can be shortened to their first two characters. Variable names are sometimes spelled oddly; this is to ensure that they don't conflict with BASIC reserved words and other variables. (In Applesoft and some other Microsoft BASICs, only the first two characters of a variable name are remembered.) In addition, try larger array sizes for the DIM (dimension) statements of both programs. Your computer may have more space available than mine does.

Listing 1 is the program I call EDITOR. With this program, you can create a new data file or modify a data file; modifications include changing elements, changing the size of the array, or expanding the array to twice its size. Listing 2, the program called RELAXN, reads this data file, does the relaxation in a semiautomated fashion, prints intermediate and final results, and enables you to save the final result onto disk for later manipulation.

The data file used by both programs has the following contents: first, the number of rows in the input array; second, the number of columns; third, a numeric value that represents an inactive node (called the "inactive number"); and, finally, the elements of the input array listed by rows.

The input array needs some explanation because it is used to represent several different kinds of data. I created the inactive number so these programs could work with rounded cross sections. The inactive number can be any value not otherwise found in the input array. You should use it only in the corners of the array to make a rounded shape fit into a rectangular array. The second kind of data is the boundary elements. The RELAXN program looks at the input array and flags the outermost layer of numbers (ignoringinactive nodes, if any) as boundary elements. The third kind of data is any elements left; these represent the initial values of the interior nodes in the cross section.

The RELAXN program reads the input array into the NODE array and, before manipulating it, creates a same-sized MASK array that stores the type of each element. Inactive elements have a MASK value of -1, boundary elements have a value of 0, and interior elements have a MASK of 1. The program checks this array often to prevent doing inappropriate operations on any given element.

The notes for listing 1 provide a commentary on the EDITOR program, which is pretty straightforward. You can start a data file from scratch or read in a previously existing one. You can change the array by row, column, or individual element. You can change the size of an array loaded in from disk and also expand an array to twice its size (actually, from m-by-n to (2m-1)-by-(2n-1)).

This option, discussed in the main text, is used to get more accurate results. When you choose to expand the input array, the computer tries to interpolate the values of added elements in a context-sensitive way. It usually does a good job, but you should inspect the resulting array and patch up any flaws.

The notes for listing 2 provide a commentary on the RELAXN program. The program reads in the input array, creates the MASK and RESID arrays, lets you do block relaxations (if desired), repeats the main loop of the iteration algorithm until the RESID array is within the specified range of accuracy, prints out the result, and allows you to save the solved NODE array for later use. The program also lets you set the number of iterations to be performed before the NODE and RESID arrays are to be printed, gives you a warning message if the relaxation "hangs" on a single node (which sometimes denotes the end of the algorithm at that level of accuracy before the official criteria for ending are fulfilled), and lets you abort the algorithm and save your results.

I wrote this program to be as simple and clear as possible; there are numerous optimizations I did not perform, leaving that to the enterprising programmer. These two programs were designed us ing a structured flowchart format I described three years ago (see "Structured Programming and Structured Flowcharts,' March 1981 , page 20). The structured flowcharts were then translated into BASIC code, whcih accounts for the occasionally unconventional use of GOTOs in the programs. I reluctantly chose BASIC over Pascal because BASIC is still the lingua franca of BYTE readers--a recent study we did showed that 77% of our readers use BASIC most often, while 21% use some kind of assembly language, and only 15% use Pascal.

One final note: I must confess to the use of a quick-and-dirty shortcut concerning the block relaxation subroutine in RELAXN. Instead of actually implementing the block relaxation algorithm (which decreases the number of computations to relax a block of elements by the same amount), I had the computer execute a double-nested do-loop that relaxed each element individually. You should implement the true block relaxation algorithm if you are going to be doing many large block relaxations. Mea culpa, mea maxima culpa.

Program Notes for Listing 1

Line Group Function

181-195 Loads in an array from disk; you have the otions of expanding the array (line 184) or arbitrarily changing its size (line 188).

200-210 Gets the size of the array (MROWS, MCOLS) and the value of the "inactive' element (INACTIVE) the array is being built from scratch.

410-465 Gets a row of values; this section repeats until you give it -1 for a row number

472-486 Gets a column of values; this section repeats until you give it -1 for a column number

505-550 Gets an individual element to change; this section repeats until you give it a -1,0,0 to end it

600-630 Saves the file to disk.

Subroutines

10000-10060 Reads data file from disk.

11000-11500 Expands array A to a twice-sized array B. Does interpolation to fill in missing values. If one of the two values used for interpolation is the inactive element, the value being interpolated is set equal to the active element.

14000-14060 Writes data file to disk.

20000-20420 Displays the array being worked on (array B). Because the array may have more columns than can be printed on an 80-column printer, I wrote this routine to display C10LPERPAGE columns at a time; I can then paste these strips together to get the entire array. You may want to

continued

change the value of C10LPERPAGE or write a more efficient, implementation-specific subroutine.

Program Notes for Listing 2

210-220 Reads input file.

300 Creates MASK array from input array.

380 Sets flag MANUAL$, which determines whether you can do point relaxations from the keyboard after every printout of the NODE and RESID arrays.

400-410 Gets the desired number of decimal places of accuracy and computes the values of two error-limit values, ERR (maximum error for any one element) and ESUM (maximum error for sum of all error values).

500 Calculates the value of the RESID array.

600-660 Enables you to do block relaxation; this section of code repeats until you answer N to the question in line 605.

800 Checks to see if relaxation algorithm is finished. If it is (very unlikely), QUIT$ is set to Y.

900 Gets the number of iterations to be performed, ITERLEFT, before the NODE and RESID arrays are printed.

1000-1300 Main loop of program, repeated until QUIT$ becomes Y. Its main events are doing a point relaxation on the element that needs it most (line 1103), and evaluating RESID for end-of-algorithm (setting QUITE$ to Y if the conditions are met--line 1200). Each time through this loop, ITERLEFT is decremented by 1; if it goes to 0, the NODE and RESID arrays are printed (line 1160), you get a chance to quit the program prematurely (if it "hangs" on certain values--line 1240), and the program gets a new value for ITERLEFT (line 1242). In certain circumstances, the variation this algorithm gives is too "coarse" to adjust the RESID array below the given error threshold. This usually results in the program relaxing the same node endlessly. The program gives you a warning if it detects that the same node has been relaxed twice in a row.

1400-1420 Gives you a chance to do manual relaxation to fine-tune the NODE array before saving it to disk.

1500-1530 Recalculates the RESID array from the NODE array and rechecks to ensure that the NODE array actually meets the terminating conditions. The program does this because each relaxation adjusts the NODE and RESID arrays, and roundoff errors may have accumulated.

2000-2110 Gives you the option to save the NODE data to a disk data file. RESID is not saved because it can be directly calculated from NODE.

Subroutines

15000-15620 Creates MASK array from input array (which is contained in the NODE array variable). All non-INACTIVE values on the first and last rows are considered to be border elements. On all other rows, the rows are inspected from the ends inward; the first non-INACTIVE value on each end is taken to be a border element. All the elements framed by the two border elements are taken to be active (interior) elements. Inactive elements are marked by 1, border elements by 0, active elements by 1 in MASK.

17000-1720 Relaxes node (I,J) by the amount N. The RESID values are changed according to the relaxation template only if the node is an active, interior

```
          one; border and inactive elements are not
          changed.
19000-19210 Does a block relaxation given the upper-left
          corner element (RLO, CLO) and lower-right corner
          element (RHI, CHI). This routine automatically
          calculates the number of units for the block to
          be relaxed in line 19080.
20000-20420 Prints the NODE and RESID arrays. See the
          reference to line 20000 in the program notes for
          listing 1.
21000-21200 Evaluates the RESID array to determine if the
          program is finished (see line 21120). If so,
          QUIT$ is set to Y.
22000-22200 Finds the element with the largest RESID value
          and relaxes it to 0.
23000-23060 Saves the NODE array and related information
          to disk.  This data file can be read again by
          either the EDITOR or RELAXN programs.
24000-24080 Enables you to do point relaxations from the
          keyboard.
```

RELX1.BAS
Contributed by: Gregg Williams
"An Introduction to Relaxation Methods," by Gregg Williams. January, page 111.

```
100  REM
110  REM    ARRAY EDITOR/EXPANDER PROGRAM
120  REM
130  REM    BY GREGG WILLIAMS, 14 NOV 83
140  REM
150  REM
155  DIM A(20,20),B(20,20)
160 QUIT =  - 1
165  PRINT : PRINT : PRINT "ARRAY EDITOR/EXPANDER PROGRAM": PRINT "BY GREGG
WILLIAMS, BYTE MAGAZINE"
167  PRINT : PRINT "THIS PROGRAM ALLOWS YOU TO CREATE AND/ORCHANGE A STARTING
ARRAY OF ELEMENTS TO  BE WORKED ON BY THE RELAXATION PROGRAM.": PRINT
170  PRINT "YOU WILL BE ABLE TO CHANGE THE ARRAY BY (IN THIS ORDER) ROWS,
COLUMNS, AND      INDIVIDUAL POINTS.": PRINT : PRINT
175  PRINT : PRINT : INPUT "LOAD STARTING ARRAY FROM DISK (Y OR N)? ";FLAG$
180  IF FLAG$ = "N" THEN  GOTO 200
181  PRINT : PRINT : INPUT "NAME OF FILE CONTAINING ARRAY?  ";NAME$
182  GOSUB 10000: REM  --INPUT FILE FROM DISK
183  GOSUB 12000: REM   --MOVE A ARRAY TO B
184  PRINT : INPUT "EXPAND ARRAY TO NEXT FINER GRID SIZE?   (Y OR N)  ";FLAG$
185  IF FLAG$ = "Y" THEN  GOSUB 11000: GOTO 188: REM --EXPAND ARRAY
187  GOSUB 12000: REM  --MOVE A ARRAY TO B
188  PRINT : PRINT : PRINT "THE ARRAY NOW LOOKS LIKE THIS: ": GOSUB 20000:
PRINT : INPUT "DO YOU WANT TO CHANGE THE SIZE OF THE   ARRAY (Y OR N)?
";ANSWER$
190  IF ANSWER$ = "Y" THEN  PRINT : INPUT "ENTER THE NEW ROW AND COLUMN SIZES:
";MROWS,MCOLS
195  GOTO 400
200  PRINT : PRINT : INPUT "NUMBER OF ROWS IN ARRAY? ";MROWS
203  PRINT : PRINT : INPUT "NUMBER OF COLUMNS IN ARRAY? ";MCOLS
205  PRINT : PRINT : PRINT "IF THE ARRAY IS NOT RECTANGULAR, YOU    WILL NEED
TO DENOTE INACTIVE GRID       POSITIONS BY A NUMBER THAT IS NOT
ELSEWHERE IN THE ARRAY."
210  PRINT : INPUT "WHAT 'INACTIVE' VALUE DO YOU WANT TO    USE (YOU HAVE TO
SUPPLY A VALUE EVEN    IF THE ARRAY IS RECTANGULAR)? ";INACTIVE
400  PRINT : PRINT : PRINT "THE ARRAY NOW LOOKS LIKE THIS: ": GOSUB 20000
405  PRINT : PRINT : PRINT "YOU CAN NOW ENTER AN ENTIRE ROW": PRINT "OF
";MCOLS;" VALUES."
410  PRINT : INPUT "ENTER ROW NUMBER TO CHANGE, OR -1       TO CONTINUE:
";ROWNUM
420  IF ROWNUM = QUIT THEN 470
430  FOR I = 1 TO MCOLS
440  PRINT "  ARRAY(";ROWNUM;",";I;")=";: INPUT B(ROWNUM,I)
450  NEXT I
460  PRINT : PRINT : PRINT "THE ARRAY NOW LOOKS LIKE THIS: ": GOSUB 20000
465  GOTO 410
```

continued

```
470  PRINT : PRINT : PRINT "THE ARRAY NOW LOOKS LIKE THIS: ": GOSUB 20000
472  PRINT : PRINT : PRINT "YOU CAN NOW ENTER AN ENTIRE COLUMN": PRINT "OF
";MROWS;" VALUES."
474  PRINT : INPUT "ENTER COLUMN NUMBER TO CHANGE, OR -1    TO CONTINUE:
";COLNUM
476  IF COLNUM = QUIT THEN 500
478  FOR I = 1 TO MROWS
480  PRINT "  ARRAY(";I;",";COLNUM;")=";: INPUT B(I,COLNUM)
482  NEXT I
484  PRINT : PRINT : PRINT "THE ARRAY NOW LOOKS LIKE THIS: ": GOSUB 20000
486  GOTO 472
500  PRINT : PRINT : PRINT "THE ARRAY NOW LOOKS LIKE THIS: ": GOSUB 20000
505  PRINT : PRINT : PRINT "YOU NOW HAVE THE OPPORTUNITY TO CHANGE  INDIVIDUAL
POINTS."
510  PRINT : INPUT "ENTER: ROW #, COLUMN #, AND NEW VALUE   TO CHANGE AN
ELEMENT OF THE ARRAY;  OR  ENTER -1,0,0 TO END AND PREPARE FOR     SAVING THE
ARRAY TO DISK:  ";ROWNUM,COLNUM,NWVLUE
520  IF ROWNUM = QUIT THEN 600
530 B(ROWNUM,COLNUM) = NWVLUE
540  PRINT : PRINT : PRINT "THE ARRAY NOW LOOKS LIKE THIS: ": GOSUB 20000
550  GOTO 510
600  PRINT : PRINT : PRINT "YOU SHOULD NOW BE FINISHED WITH THE     ARRAY.  ";
610  INPUT "UNDER WHAT FILENAME DO YOU WANT TO SAVE IT? ";NAME$
620  GOSUB 14000
630  PRINT : PRINT : PRINT "FILE ";NAME$;" SAVED": PRINT "END OF PROGRAM"
640  END
9990  REM
9992  REM
9994  REM       READ FROM FILE
9995  REM       NAME$ INTO ARRAY A
9996  REM
9998  REM
10000 D$ =  CHR$ (13) +  CHR$ (4)
10005  PRINT D$;"OPEN ";NAME$
10010  PRINT D$;"READ ";NAME$
10020  INPUT MROWS
10030  INPUT MCOLS
10040  INPUT INACTIVE
10050  FOR I = 1 TO MROWS: FOR J = 1 TO MCOLS: INPUT A(I,J): NEXT J: NEXT I
10055  PRINT D$;"CLOSE ";NAME$
10057  PRINT : PRINT "FILE ";NAME$;" READ FROM DISK.": PRINT "IT IS A
";MROWS;" BY ";MCOLS;" ARRAY."
10058  PRINT "INACTIVE ELEMENTS ARE DENOTED BY ";INACTIVE;"."
10060  RETURN
10990  REM
10992  REM
10994  REM       EXPAND A TO TWICE-
10995  REM       SIZED ARRAY B
10996  REM
10998  REM
11000  PRINT : PRINT : PRINT "THE INPUT ARRAY IS: "
11006  REM  --AT THIS POINT, ARRAY A HAS BEEN COPIED
11008  REM  --TO B; THE FOLLOWING PRINTS ARRAY B
11010  GOSUB 20000
11092  REM
11094  REM  --CLEAR B ARRAY WITH INACTIVE VALUE
11096  REM
11098  FOR I = 1 TO 2 * MROWS - 1: FOR J = 1 TO 2 * MCOLS - 1:B(I,J) =
INACTIVE: NEXT J: NEXT I
11100  REM
11110  REM  --EXPLODE A, CREATING B
11120  REM
11130  FOR I = 1 TO MROWS: FOR J = 1 TO MCOLS
11140 B(2 * I - 1,2 * J - 1) = A(I,J)
11150  NEXT J: NEXT I
11160  REM
11170  REM  --INTERPOLATE VALUES FOR COLUMNS 1, 3, 5, ...
11180  REM
11200  FOR J = 1 TO 2 * MCOLS - 1 STEP 2: FOR I = 2 TO 2 * MROWS - 2 STEP 2
11210  IF B(I - 1,J) = INACTIVE THEN B(I,J) = B(I + 1,J): GOTO 11240
11220  IF B(I + 1,J) = INACTIVE THEN B(I,J) = B(I - 1,J): GOTO 11240
11230 B(I,J) =  INT ((B(I - 1,J) + B(I + 1,J)) / 2 + 0.5)
11240  NEXT I: NEXT J
11250  REM
11260  REM  --INTERPOLATE VALUES FOR COLUMNS 2, 4, 6, ...
11270  REM
```

```
11300  FOR J = 2 TO 2 * MCOLS - 2 STEP 2: FOR I = 1 TO 2 * MROWS - 1
11310  IF B(I,J - 1) = INACTIVE THEN B(I,J) = B(I,J + 1): GOTO 11340
11320  IF B(I,J + 1) = INACTIVE THEN B(I,J) = B(I,J - 1): GOTO 11340
11330 B(I,J) =  INT ((B(I,J - 1) + B(I,J + 1)) / 2 + 0.5)
11340  NEXT I: NEXT J
11345 MROWS = 2 * MROWS - 1:MCOLS = 2 * MCOLS - 1
11350  REM
11500  RETURN
11990  REM
11992  REM
11994  REM       COPY A TO B
11996  REM
11998  REM
12000  FOR I = 1 TO MROWS: FOR J = 1 TO MCOLS:B(I,J) = A(I,J): NEXT J: NEXT I:
RETURN
13990  REM
13992  REM
13994  REM         SAVE ARRAY B
13996  REM         TO FILE $NAME
13998  REM
14000 D$ =  CHR$ (13) +  CHR$ (4)
14005  PRINT D$;"OPEN ";NAME$
14010  PRINT D$;"WRITE ";NAME$
14020  PRINT MROWS
14030  PRINT MCOLS
14040  PRINT INACTIVE
14050  FOR I = 1 TO MROWS: FOR J = 1 TO MCOLS: PRINT B(I,J): NEXT J: NEXT I
14055  PRINT D$;"CLOSE ";NAME$
14060  RETURN
19990  REM
19992  REM
19994  REM       DISPLAY B ARRAY
19996  REM
19998  REM
20000 C1OLPERPAGE = 5:L1IMLOW = 1
20010  IF L1IMLOW > MCOLS THEN 20150
20014  REM     --C2DIFFERENCE=MIN OF C1OLPERPAGE AND (MCOL-L1IMLOW+1)
20015 C2DFFERENCE = MCOL - L1IMLOW + 1
20017  IF C1OLPERPAGE < C2DFFERENCE THEN C2DFFERENCE = C1OLPERPAGE
20018 J1 = L1IMLOW
20020 J2 = L1IMLOW + C2DFFERENCE - 1
20030  PRINT : PRINT : PRINT "ARRAY IS:"
20080  GOSUB 20400
20090  GOSUB 20200
20095  REM
20100 L1IMLOW = L1IMLOW + C2DFFERENCE
20110  GOTO 20010
20150  PRINT
20170  RETURN
20199  REM
20200  FOR I = 1 TO MROWS: FOR J = J1 TO J2
20210  PRINT B(I,J),
20220  NEXT J: PRINT : NEXT I
20230  RETURN
20299  REM
20400  PRINT
20410  FOR J = J1 TO J2: PRINT "COL # ";J,: NEXT J: PRINT
20420  RETURN
```

RELX2.BAS
Contributed by: Gregg Williams
"An Introduction to Relaxation Methods," by Gregg Williams. January, page 111.

```
8  REM
60  REM
65  REM    ARRAY RELAXATION PROGRAM
70  REM
75  REM    BY GREGG WILLIAMS, 28 NOV 83
80  REM
85  REM
100  PRINT : PRINT : PRINT "TWO-DIMENSIONAL RELAXATION ALGORITHM": PRINT "BY
GREGG WILLIAMS, BYTE MAGAZINE": PRINT : PRINT
```

continued

```
110  PRINT "THIS PROGRAM READS AN INPUT ARRAY FROM  A DISK FILE, ALLOWS YOU TO
DO (OPTIONAL)BLOCK RELAXATIONS, THEN AUTOMATICALLY   GOES THROUGH THE PROCESS
OF FINDING A   SOLUTION VIA RELAXATION.": PRINT
120  PRINT "YOU CAN TELL THIS PROGRAM HOW OFTEN YOU WANT TO LOOK AT THE
INTERMEDIATE        RESULTS AND WHEN YOU WANT TO BE ABLE TO DO POINT
RELAXATION MANUALLY.  YOU CAN"
130  PRINT "ABORT THE PROGRAM AFTER ANY PRINTOUT    AND SAVE THE RESULTING
ARRAY, FINISHED  OR NOT, TO DISK."
140  PRINT : PRINT : PRINT
150  REM
152  REM
154  REM      HOUSEKEEPING
156  REM
158  REM
160 QUIT =  - 1
162  DIM NODE(20,20),MASK(20,20),RESID(20,20)
165 COUNTITERS = 0: REM  --KEEPS TRACK OF # OF ITERATIONS COMPUTER HAS DONE
170 LCOL = 0:LROW = 0: REM  --THESE WILL KEEP COORDINATES OF PREVIOUS
RELAXATION
190  REM
192  REM
194  REM      READ INPUT FILE
196  REM
198  REM
210  INPUT "WHAT INPUT FILE DO YOU WANT TO USE?     ";NAME$
220  GOSUB 10000: REM   --READ MROWS, MCOLS, INACTIVE, NODE ARRAY
290  REM
292  REM
294  REM      CREATE MASK ARRAY
295  REM      FROM NODE ARRAY
296  REM
298  REM
300  GOSUB 15000
370  REM
372  REM
374  REM      DO MANUAL RELAX'N
375  REM      AFTER PRINTING?
376  REM
378  REM
380  PRINT : PRINT : INPUT "DO YOU WANT TO DO MANUAL RELAXATION ON  INDIVIDUAL
POINTS AFTER YOU SEE THE     ARRAYS ARE DISPLAYED (Y OR N)? ";MANUAL$
390  REM
392  REM
394  REM      GET DECIMAL PLACES
395  REM      OF ACCURACY
396  REM
398  REM
400  PRINT : PRINT : INPUT "ENTER THE NUMBER OF DECIMAL PLACES OF   ACCURACY
DESIRED FOR THE CALCULATIONS:  ";S1CALE
405 S1CALE = S1CALE + 1
410 ERR = 5 * 10 ^  - S1CALE:ESUM = 10 ^ ( - S1CALE + 1)
490  REM
492  REM
494  REM      CALCULATE RESID
495  REM      ARRAY
496  REM
498  REM
500  GOSUB 18000
590  REM
592  REM
594  REM      DO BLOCK RELAXATION
596  REM
598  REM
600  GOSUB 20000: REM    --DISPLAY NODE AND RESID
605  PRINT : PRINT : INPUT "DO YOU WANT TO DO A BLOCK RELAXATION    (Y OR N)?
";QUIT$
610  IF QUIT$ = "N" THEN 800
612 GOOD$ = "F"
614  IF GOOD$ = "T" THEN 640
615  PRINT : PRINT : PRINT "THE UPPER LEFT CORNER IS ELEMENT (1,1)  (I.E., ROW
1, COLUMN 1), AND THE LOWER  RIGHT ELEMENT IS (";MROWS;",";MCOLS;")."
620  PRINT : INPUT "GIVE THE ROW AND COLUMN NUMBER OF THE   UPPER LEFT CORNER
OF THE BLOCK RELAX-   ATION: ";RLO,CLO
630  PRINT : INPUT "GIVE THE ROW AND COLUMN NUMBER OF THE   LOWER RIGHT CORNER
OF THE BLOCK RELAX-  ATION: ";RHI,CHI
635  IF MASK(RLO,CLO) = 1 AND MASK(RHI,CHI) = 1 THEN GOOD$ = "T"
```

```
636  IF GOOD$ = "F" THEN  PRINT : PRINT "AT LEAST ONE POINT SPECIFIED IS NOT
AN  INTERIOR NODE--TRY AGAIN"
637  GOTO 614
640  GOSUB 19000
650  GOSUB 20000: REM  --DISPLAY NODE AND RESID
660  GOTO 605
790  REM
792  REM
794  REM      EVALUATE RESIDUALS
796  REM
798  REM
800  GOSUB 21000
890  REM
892  REM
894  REM      GET # OF ITERATIONS
895  REM      UNTIL NEXT PRINTOUT
896  REM
898  REM
900  PRINT : PRINT : INPUT "HOW MANY ITERATIONS DO YOU WANT TO DO   BEFORE THE
NODE AND RESID ARRAYS ARE    PRINTED? ";ITERLEFT
980  REM
990  REM
992  REM    **************
994  REM    * MAIN LOOP  *
995  REM    * OF PROGRAM *
996  REM    **************
998  REM
1000  IF QUIT$ = "Y" THEN 1400
1090  REM
1092  REM
1094  REM      DO RELAXATION ON
1096  REM      LARGEST RESIDUAL
1097  REM
1098  REM
1100  PRINT : PRINT : PRINT "--STARTING RELAXATION #";COUNTITERS + 1;"--"
1103  GOSUB 22000
1105 ITERLEFT = ITERLEFT - 1
1107 COUNTITERS = COUNTITERS + 1
1110  REM
1120 DUPLICN$ = "N"
1125  IF LROW = RROW AND LCOL = RCOL THEN DUPLICN$ = "Y"
1130  IF ITERLEFT > 0 THEN 1250
1140  REM
1150  REM  --THIS DONE IF WE ARE PRINTING RESULTS
1160  GOSUB 20000: REM  --PRINT ARRAYS
1170  REM
1174  REM  --POINT RELAXATION BY HUMAN (OPTIONAL)
1175  IF MANUAL$ = "Y" THEN  GOSUB 24000
1180  REM
1185  IF DUPLICN$ = "N" THEN 1200
1190  PRINT : PRINT "THE PROGRAM HAS RELAXED ON THE SAME     POINT TWICE.
THIS PROBABLY MEANS THAT  THIS PROGRAM WILL NEVER END": PRINT : PRINT "  ***
YOU MAY WANT TO QUIT ***"
1197  PRINT : INPUT "NUMBER OF ITERATIONS UNTIL NEXT         PRINTOUT?
";MXITERS:ITERLEFT = MXITERS
1198  REM
1200  GOSUB 21000: REM  --EVALUATE RESID, UPDATE QUIT$
1220  IF QUIT$ = "Y" THEN 1300
1240  PRINT : INPUT "DO YOU WANT TO QUIT CALCULATIONS AND    SAVE THE NODE
ARRAY AS IS (Y OR N)? ";QUIT$
1241  REM
1242  PRINT : PRINT : INPUT "HOW MANY ITERATIONS DO YOU WANT TO DO   BEFORE
THE NODE AND RESID ARRAYS ARE    PRINTED? ";ITERLEFT
1243  REM
1245  GOTO 1300
1248  REM
1249  REM  --THIS DONE IF WE ARE NOT PRINTING RESULTS
1250  GOSUB 21000: REM   --EVALUATE RESID, UPDATE QUIT$
1260  IF DUPLICN$ = "N" THEN 1300
1270  PRINT : PRINT "THE PROGRAM HAS JUST RELAXED ON THE     SAME POINT TWICE.
THIS PROBABLY MEANS  THAT THIS PROGRAM WILL NEVER END.": PRINT
1280  PRINT "  *** YOU MAY WANT TO QUIT THIS ***       *** PROGRAM AT THE NEXT
***      *** OPPORTUNITY                  ***"
1296  REM
1298  REM
```

continued

```
1300  GOTO 1000: REM  --END 'WHILE' LOOP
1308  REM
1310  REM
1312  REM       **************
1314  REM       *   END OF   *
1316  REM       * MAIN LOOP  *
1318  REM       **************
1320  REM
1322  REM
1390  REM
1392  REM
1395  REM      DO MANUAL
1396  REM      RELAXATION
1397  REM
1399  REM
1400  PRINT : PRINT : INPUT "DO YOU WANT TO DO ANY POINT RELAXATIONS MANUALLY
BEFORE ENDING THIS PROGRAM? (Y OR N)? ";FLAG$
1420  IF FLAG$ = "Y" THEN  GOSUB 24000
1492  REM
1494  REM      RECALCULATE RESULTS
1495  REM      FOR ACCURACY
1496  REM
1498  REM
1500  GOSUB 18000: REM  --RECALCULATE RESID ARRAY
1510  GOSUB 21000: REM  --EVALUATE RESID ARRAY
1520  GOSUB 20000: REM  --DISPLAY NODE, RESID, BIGRESID, SUMRESID
1530  PRINT : PRINT "THIS PROGRAM PERFORMED ";COUNTITERS;" AUTOMATIC": PRINT
"POINT RELAXATIONS"
1990  REM
1992  REM
1994  REM      WRITE FILE TO
1995  REM      DISK (OPTIONAL)
1996  REM
1998  REM
2000  PRINT : PRINT : INPUT "THIS PROGRAM HAS FINISHED ITS WORK.  DO YOU WANT
TO SAVE THE NODE ARRAY          (Y OR N)?  ";FLAG$
2010  IF FLAG$ = "N" THEN 2100
2020  REM
2030  PRINT : INPUT "UNDER WHAT FILENAME DO YOU WANT TO SAVE IT? ";NAME$
2040  GOSUB 23000: REM  --SAVE WORK TO DISK
2050  PRINT : PRINT : PRINT "FILE ";NAME$;" SAVED": PRINT "END OF PROGRAM"
2070  PRINT  CHR$ (4);"PR#0"
2100  END
2102  REM  --EVALUATE RESID ONLY IF RELAXN IS TO CONTINUE
2110  PRINT : PRINT : PRINT "FILE ";NAME$;" SAVED": PRINT "END OF PROGRAM"
9980  REM
9982  REM  *********************
9984  REM  *                   *
9985  REM  * END MAIN PROGRAM, *
9986  REM  * BEGIN SUBROUTINES *
9987  REM  *                   *
9988  REM  *********************
9990  REM
9992  REM
9994  REM      READ FROM FILE
9995  REM      NAME$ INTO ARRAY A
9996  REM
9998  REM
10000 D$ =  CHR$ (13) +  CHR$ (4)
10005  PRINT D$;"OPEN ";NAME$
10010  PRINT D$;"READ ";NAME$
10020  INPUT MROWS
10030  INPUT MCOLS
10040  INPUT INACTIVE
10050  FOR I = 1 TO MROWS: FOR J = 1 TO MCOLS: INPUT NODE(I,J): NEXT J: NEXT I
10055  PRINT D$;"CLOSE ";NAME$
10057  PRINT "FILE ";NAME$;" READ FROM DISK"
10060  RETURN
14990  REM
14992  REM
14994  REM       CREATE MASK ARRAY
14995  REM       FROM NODE ARRAY
14996  REM
15000  FOR I = 1 TO MROWS STEP (MROWS - 1)
15010  REM  --DOES LOOP TWICE: I=1, I=MROWS
15020  REM
```

```
15030  FOR J = 1 TO MCOLS
15035  REM   --CALCULATE FIRST AND LAST ROWS
15040  REM
15050  IF NODE(I,J) = INACTIVE THEN MASK(I,J) =  - 1: GOTO 15100
15060 MASK(I,J) = 0: REM  --HERE, MUST BE BORDER ELEMENT
15090  REM
15100  NEXT J: NEXT I
15190  REM
15200  FOR I = 2 TO (MROWS - 1)
15210  REM  --DO ALL ROWS EXCEPT FIRST AND LAST
15220  REM
15230  REM  --ATTACK ROW FROM EACH END; FIRST
15240  REM    VALUE <> INACTIVE IS BORDER (MASK=0);
15250  REM    NEXT VALUE IS ACTIVE (MASK=1); WORK
15260  REM    SIMILARLY FROM END OF ROW
15270  REM
15280 LOCOL = 1:HICOL = MCOLS
15290  REM
15300  IF NODE(I,LOCOL) <  > INACTIVE THEN 15400
15310 MASK(I,LOCOL) =  - 1:LOCOL = LOCOL + 1
15320  GOTO 15300
15390  REM
15400  IF NODE(I,HICOL) <  > INACTIVE THEN 15500
15410 MASK(I,HICOL) =  - 1:HICOL = HICOL - 1
15420  GOTO 15400
15490  REM
15500 MASK(I,LOCOL) = 0:LOCOL = LOCOL + 1
15510 MASK(I,HICOL) = 0:HICOL = HICOL - 1
15520  REM
15530  IF LOCOL >  = HICOL THEN 15600
15540  REM
15550  FOR J = LOCOL TO HICOL
15560 MASK(I,J) = 1
15570  NEXT J
15590  REM
15600  NEXT I
15610  REM
15620  RETURN
16990  REM
16992  REM
16994  REM      RELAX NODE (I,J)
16995  REM      BY N
16996  REM
16998  REM
17000  IF MASK(I - 1,J) <  > 1 THEN 17020
17010 RESID(I - 1,J) = RESID(I - 1,J) + N
17020  IF MASK(I + 1,J) <  > 1 THEN 17040
17030 RESID(I + 1,J) = RESID(I + 1,J) + N
17040  IF MASK(I,J - 1) <  > 1 THEN 17060
17050 RESID(I,J - 1) = RESID(I,J - 1) + N
17060  IF MASK(I,J + 1) <  > 1 THEN 17100
17070 RESID(I,J + 1) = RESID(I,J + 1) + N
17090  REM
17100 RESID(I,J) = RESID(I,J) - 4 * N
17110 NODE(I,J) = NODE(I,J) + N
17200  RETURN
17990  REM
17992  REM
17994  REM      CALCULATE RESID
17995  REM      ARRAY
17996  REM
17998  REM
18000  FOR I = 1 TO MROWS: FOR J = 1 TO MCOLS
18010 RESID(I,J) = 0
18020  NEXT J: NEXT I
18030  FOR I = 2 TO MROWS - 1: FOR J = 2 TO MCOLS - 1
18040  IF MASK(I,J) <  > 1 THEN 18100
18050 RESID(I,J) = NODE(I - 1,J) + NODE(I + 1,J) + NODE(I,J - 1) + NODE(I,J +
1) - 4 * NODE(I,J)
18100  NEXT J: NEXT I
18110  RETURN
18990  REM
18992  REM
18994  REM      DO BLOCK RELAXATION
18996  REM
```

continued

```
18998  REM
19000 RSUM = 0
19010  FOR I = RLO TO RHI: FOR J = CLO TO CHI
19020 RSUM = RSUM + RESID(I,J)
19030  NEXT J: NEXT I
19040 RTEMP = RSUM
19045 BLOCKLEADS = 2 * (RHI - RLO + 1) + 2 * (CHI - CLO + 1)
19070  REM
19080 RADJUST = RSUM / BLOCKLEADS:N = RADJUST
19090  REM
19100  FOR I = RLO TO RHI: FOR J = CLO TO CHI
19110  GOSUB 17000: REM  --RELAX BY N=RADJUST
19120  NEXT J: NEXT I
19190  REM
19200  PRINT : PRINT : PRINT "BLOCK RELAXATION FINISHED": PRINT : PRINT "  SUM
IS ";RTEMP: PRINT "  BLOCK ADJUSTED BY ";RADJUST;" UNITS"
19210  RETURN
19990  REM
19992  REM
19994  REM      PRINT NODE, RESID
19995  REM       ARRAYS
19996  REM
19998  REM
20000 C1OLPERPAGE = 5:L1IMLOW = 1
20010  IF L1IMLOW > MCOL THEN 20150
20014  REM    --C2DIFFERENCE=MIN OF C1OLPERPAGE AND (MCOL-L1IMLOW+1)
20015 C2DFFERENCE = MCOL - L1IMLOW + 1
20017  IF C1OLPERPAGE < C2DFFERENCE THEN C2DFFERENCE = C1OLPERPAGE
20018 J1 = L1IMLOW
20020 J2 = L1IMLOW + C2DFFERENCE - 1
20030  PRINT : PRINT : PRINT "NODE ARRAY IS:"
20040  GOSUB 20400
20050  GOSUB 20300
20060  REM
20070  PRINT : PRINT : PRINT "RESID ARRAY IS:"
20080  GOSUB 20400
20090  GOSUB 20200
20095  REM
20100 L1IMLOW = L1IMLOW + C2DFFERENCE
20110  GOTO 20010
20150  PRINT
20170  RETURN
20199  REM
20200  FOR I = 1 TO MROWS: FOR J = J1 TO J2
20210  PRINT  INT (10 ^ S1CALE * RESID(I,J) + 0.5) / 10 ^ S1CALE,
20220  NEXT J: PRINT : NEXT I
20230  RETURN
20299  REM
20300  FOR I = 1 TO MROWS: FOR J = J1 TO J2
20310  PRINT  INT (10 ^ S1CALE * NODE(I,J) + 0.5) / 10 ^ S1CALE,
20320  NEXT J: PRINT : NEXT I
20330  RETURN
20399  REM
20400  PRINT
20410  FOR J = J1 TO J2: PRINT "COL # ";J,: NEXT J: PRINT
20420  RETURN
20990  REM
20992  REM
20994  REM      EVALUATE RESID ARRAY
20995  REM      FOR END OF RELAXATION
20996  REM
20998  REM
21000 RSUM = 0:RMAX =  ABS (RESID(2,2)):QUIT$ = "N"
21007 RSUM = 0:RMAX =  ABS (RESID(2,2)):QUIT$ = "N"
21010  FOR I = 2 TO MROWS - 1: FOR J = 2 TO MCOLS - 1
21015  IF MASK(I,J) <  > 1 THEN 21100
21020 RSUM = RSUM + RESID(I,J)
21030  IF RMAX <  ABS (RESID(I,J)) THEN RMAX =  ABS (RESID(I,J))
21090  REM
21100  NEXT J: NEXT I
21105  PRINT : PRINT "RESIDUAL SUM = ";RSUM
21110  PRINT "LARGEST ABSOLUTE VALUE = ";RMAX
21120  IF RMAX > ERR OR  ABS (RSUM) > ESUM THEN 21150
21130  PRINT : PRINT "ARRAY WITHIN TOLERANCES--RELAXATION IS  FINISHED":QUIT$
= "Y": GOTO 21200
21150  PRINT : PRINT "ARRAY NOT WITHIN TOLERENCES--CONTINUING"
21200  RETURN
```

```
21990  REM
21992  REM
21994  REM      RELAX ON NODE
21995  REM      W/ LARGEST RESID
21996  REM
21998  REM
22000 LROW = RROW:LCOL = RCOL
22005 RROW = 2:RCOL = 2:RMAX =  ABS (RESID(RROW,RCOL))
22010  FOR I = 2 TO MROWS - 1: FOR J = 2 TO MCOLS - 1
22020  REM
22030  IF MASK(I,J) <  > 1 OR RMAX >  =  ABS (RESID(I,J)) THEN 22100
22040  REM
22050 RMAX =  ABS (RESID(I,J)):RROW = I:RCOL = J
22060  REM
22100  NEXT J: NEXT I
22105 RADJUST = RESID(RROW,RCOL) / 4
22135  IF RADJUST = 0 THEN RADJUST =  SGN (RESID(RROW,RCOL))
22140  PRINT : PRINT : PRINT "NODE(";RROW;",";RCOL;"), WITH VALUE
";RESID(RROW,RCOL);", IS RELAXED BY ";RADJUST
22145  REM
22150  REM  --SETUP FOR SUBROUTINE
22160 N = RADJUST:I = RROW:J = RCOL
22170  GOSUB 17000: REM   --RELAX-BY-N SUBROUTINE
22200  RETURN
22990  REM
22992  REM
22994  REM         SAVE NODE ARRAY
22996  REM         TO FILE $NAME
22998  REM
23000  PRINT  CHR$ (4);"OPEN ";NAME$
23010  PRINT  CHR$ (4);"WRITE ";NAME$
23020  PRINT MROWS
23030  PRINT MCOLS
23040  PRINT INACTIVE
23050  FOR I = 1 TO MROWS: FOR J = 1 TO MCOLS: PRINT NODE(I,J): NEXT J: NEXT I
23055  PRINT  CHR$ (4);"CLOSE ";NAME$
23060  RETURN
23990  REM
23992  REM
23995  REM      DO MANUAL
23996  REM      RELAXATION
23997  REM
23999  REM
24000 Q2UIT$ = "N"
24010  IF Q2UIT$ = "Y" THEN 24080
24020  PRINT : PRINT "ENTER THE ROW #, COLUMN #, AND VALUE BY WHICH THIS POINT
WILL BE RELAXED (OR -1,0,0 TO QUIT): ";
24030  INPUT I,J,N
24040  IF I =  - 1 THEN Q2UIT$ = "Y": GOTO 24070
24050  GOSUB 17000: REM  --RELAX POINT BY N
24060  GOSUB 20000: REM  --PRINT NODE, RESID ARRAYS
24070  GOTO 24010
24080  RETURN
```

SAVERGN.ASM
Contributed by: Howard Katz
"Region Maker," by Howard Katz. January, page 145.

```
; SaveRgn.ASM   20 July '86  h. katz

; Abstracted from < RgnMaker.ASM >

; This module handles the code for creating a Macintosh resource from
; the region in the Work Area which was produced by the contouring
; algorithm in < Traverse.ASM >.

; It uses a Dialog Box to get the user-input parameters for defining
; the Resource Type, ID, and Name, reloading these DITLs from memory if a
; resource has already been written to disk this session.

; The program uses the _Pack3 SFPutFile routine to get the name of
; the file the user wishes to add the resource to.
```

continued

```
INCLUDE MacTraps.D

XDEF    Save_To_File, Have_Prior_DITLs  ; defined here but
XDEF    ItemHit, PutFile_Posit          ; used in < RgnMaker.ASM >

XREF    Save_Image, Restore_Image, Stop_Alert, RgnHandle, UnHiLite

SFPutFile               equ     1       ; selector for _Pack3
PutFile_ID              equ     -3999
Replace_DITL            equ     -3996

ioNamePtr               equ     18
ioVRefNum               equ     22

StringToNum             equ     1

Res_DLOG_ID             equ     100
Cancel_Button           equ     2
Res_Type_DITL_Num       equ     7
Res_ID_DITL_Num         equ     8
Res_Name_DITL_Num       equ     9


Save_To_File

        _InitCursor

        move    #Res_DLOG_ID, d3        ; ResID of upcoming Dialog
        move.l  #'DLOG', a3
        BSR     Save_Image              ; save Bits to be hidden by the DLOG

        bsr     Open_Res_DLOG
        tst.b   Have_Prior_DITLs(a5)    ; did we do this before ?
        BEQ.s   Get_User_Choice         ; no - first time

        BSR     Reload_Old_DITLs

Get_User_Choice

        st      Have_Prior_DITLs(a5)

        BSR     Modal_DLOG              ; get kbd-input for Resource Type,
        tst     d4                      ; ID, and Name
        bmi.s   Save_To_File            ; input param error - try again

        pea     Res_DLOG_Storage        ; all done - dispose of Dialog
        _CloseDialog                    ; (and erase from screen)

        BSR     Restore_Image           ; redraw what was underneath

        cmp     #Cancel_Button, d4
        beq     UnHiLite                ; User selected 'Cancel'

        BSR     Put_To_File             ; everything OK - SFPutFile

        BRA     UnHiLite

Reload_Old_DITLs        ; restore Dialog Items saved in prior session

        move    #Res_Type_DITL_Num, D4
        BSR     Get_Item_Handle
        move.l  ItemHandle, -(sp)
        pea     Resource_Type_STR
        _SetIText
        move    #Res_Id_DITL_Num, D4
        BSR     Get_Item_Handle
        move.l  ItemHandle, -(sp)
        pea     Resource_ID_Str
        _SetIText

        move    #Res_Name_DITL_Num, D4
        BSR     Get_Item_Handle
        move.l  ItemHandle, -(sp)
        pea     Resource_Name
        _SetIText
        RTS


Get_Item_Handle

        pea     Res_DLOG_Storage        ; DLOG Ptr
        move    d4, -(sp)               ; Item Number
        pea     ItemType                ; not used
        pea     ItemHandle              ; passed to following ROM call
        pea     ItemBox                 ; not used
        _GetDItem

        RTS


Put_To_File

; The user has successfully specified the Resource TYPE and ID.
; Append it to an existing file, or create a new one if necessary

; The 1st piece of code replaces the message 'Replace Existing ...
; with an 'Are You Sure ?', which makes more sense in the
; circumstances, as we're not replacing anything.


        clr.l   -(sp)
        move.l  #'DITL', -(sp)
        move    #Replace_DITL, -(sp)    ; 'Replace Existing File' DITL
        _GetResource
        move.l  (sp)+, a0               ; get the Handle

        cmpa.l  #0, a0
        BEQ.s   @SFPutFile              ; couldn't get DITL - forget it

        move.l  (a0), a0                ; get the Ptr to DITL data
        move    -2(a0), d0              ; Block size ( in Block Header )

@Search_Length_Byte

        cmp.b   #$17, (a0)+             ; look for matching String Length
        dBEQ    d0, @Search_Length_Byte
        cmp     #-1, d0
        BEQ.s   @SFPutFile              ; fell thru without finding match

        cmp.b   #'R', (a0)              ; check for 'R' of 'Replace'
        BNE.s   @SFPutFile              ; forget it

        lea     'Are You Sure ?         ', a1

        move.b  (a1)+, d1               ; save Length Byte
        ext.w   d1                      ; get rid of what was in hi nibble
        sub     #1, d1

@Replace_Str

        move.b  (a1)+, (a0)+
        DBRA    d1, @Replace_Str

@SFPutFile

        move    #PutFile_ID, d3         ; ResID of upcoming Dialog
        move.l  #'DLOG', a3
        BSR     Save_Image              ; save Bits to be hidden by the DLOG

        move.l  PutFile_Posit, -(sp)    ; Global TopLeft of Dialog
        pea     'Append Resource to File'
        pea     'Resources'
        move.l  #0, -(sp)
        pea     SFReply
        move    #SFPutFile, -(sp)
        _Pack3

        BSR     Restore_Image           ; redraw what was underneath
        move.b  good, d0
        tst.b   d0                          ; Did user 'Cancel' out ?
        beq     @Rts                        ; yes

        BSR     Create_ioPB
        _GetVol                             ; get Info on the Default Volume

        move    iovRefNum(a0), Default_Vol(a5)  ; save the RefNum
        move    iovRefNum(a0), d0               ; to restore it later
        cmp     vRefNum, d0                 ; is file we want to add resource to
                                            ; on the Default Volume ?
        beq.s   @1                          ; yes

        clr.l   ioNamePtr(a0)               ; no
        move    vRefNum, iovRefNum(a0)
        _SetVol                             ; make it the Default Volume

@1      add.l   d4, sp                      ; clear the ioPB off the stack

        clr     -(sp)
        _CurResFile
        move    (sp)+, CurResFile(a5)       ; save the refNum for later

        clr     -(sp)
        pea     fName                       ; File Name that user entered
        _OpenResFile                        ; try to open it
        move    (sp)+, d3                   ; save the refNum for later

        clr     -(sp)
        _ResError                           ; see if we could open it
        move    (sp)+, d0
        tst     d0
        beq.s   @2                          ; we could

        pea     fName                       ; we couldn't
        _CreateResFile                      ; so try to create it
        clr     -(sp)
        _ResError
        move    (sp)+, d0
        tst     d0
        bne     @9                          ; we couldn't create it

        clr     -(sp)                       ; we created it
        pea     fName                       ; now try to open it
        _OpenResFile
        move    (sp)+, d3                   ; save the refNum for later
        clr     -(sp)
        _ResError                           ; check if we were able to open
        move    (sp)+, d0
        tst     d0

        beq.s   @2                          ; hunky-dory

        move    #105, d3                    ; 'Can't add to file' message
        BSR     Stop_Alert
        bra     @9

@2      move    d3, -(sp)                   ; we were able to open the specified
file
        _UseResFile                         ; make it the current Resource File

        clr.l   -(sp)                       ; check to see if we have a duplicate
        move.l  Resource_Type, -(sp)        ; push Type (w/out Length Byte)
        move.l  Resource_ID, d0             ; push low word of ID
        move    d0, -(sp)
        _GetResource

        move.l  (sp)+, d0                   ; check the returned Handle
        beq.s   @3                          ; NIL Handle = OK, no Duplicate

        clr     -(sp)                       ; space for INT result
        move.l  d0, -(sp)                   ; push handle again
        _HomeResFile
        move    (sp)+, d0                   ; get refNum
        cmp     d0, d3                      ; is it in the currResFile we're using
?
```

```
        bne.s   @3                      ; no = OK

        move    #104, d3                ; alertID for 'Duplicate Resource'
        BSR     Stop_Alert
        bra     @5                      ; detach the resource

@3      move.l  RgnHandle(a5), -(sp)    ; point to the Region ( theData )
        move.l  Resource_Type, -(sp)
        move.l  Resource_ID, d0         ; ID saved initially as a LONG
        move    d0, -(sp)               ; push low word only
        pea     Resource_Name
        _AddResource

        clr     -(sp)                   ; check for errors
        _ResError
        move    (sp)+, d0               ; get the result
        beq.s   @4                      ; no error

        move    #106, d3                ; 'Can't Add to File' AlertID
        BSR     Stop_Alert
        bra     @9

@4      move.l  RgnHandle(a5), -(sp)
        _WriteResource
        clr     -(sp)                   ; check for errors
        _ResError
        move    (sp)+, d0
        beq.s   @5                      ; no error

        move    #105, d3                ; 'Can't Add to File' AlertID
        BSR     Stop_Alert
        bra     @9

@5      move.l  RgnHandle(a5), -(sp)    ; just a region again
        _DetachResource                 ; (no longer a resource)

@9      BSR     Create_ioPB             ; restore original Default Volume
        move    Default_Vol(a5), iovRefNum(a0)
        _SetVol
        add.l   d4, sp                  ; clear ioPB off the stack

        move    CurResFile(a5), -(sp)
        _UseResFile                     ; restore prior Resource File

@Rts    RTS


Create_ioPB

        move.l  (sp)+, a1
        move.l  #100, d0        ; set up a temp ioPB on the stack
        move.l  d0, d4          ; save for cleanup
        asr     #2, d0          ; 100 Bytes -> 25 Longs (more than we really
        sub     #1, d0          ; need, actually)

@Push
        move.l  #0, -(sp)
        dbra    d0, @Push

        move.l  sp, a0          ; sp = addr of ioParamBlock
        jmp     (a1)


Open_Res_DLOG

        clr.l   -(sp)                   ; space for funct result
        move    #Res_DLOG_ID, -(sp)
        pea     Res_DLOG_Storage
        move.l  #-1, -(sp)              ; in front of everything
        _GetNewDialog
        move.l  (sp)+, d0
        RTS


Modal_DLOG
```

continued

```
        clr.l   -(sp)           ; no filterProc
        pea     ItemHit
        _ModalDialog
        move    ItemHit, d4
        cmp     #1, d4          ; is it 'OK' ?
        beq.s   @OK_Button      ; yes - save DITL EditText Items
        cmp     #2, d4          ; is it 'Cancel' ?
        BNE     Modal_DLOG      ; not yet - stick around

        RTS

@OK_Button

        move    #Res_Type_DITL_Num, d3
        bsr     Save_Text

        lea     Resource_Type_Str, a0
        move.b  (a0)+, d0               ; also save the Resource Type
        lea     Resource_Type, a1       ; w/o a Length Byte for _AddResource
        move.b  (a0)+, (a1)+
        move.b  (a0)+, (a1)+
        move.b  (a0)+, (a1)+
        move.b  (a0)+, (a1)+

        move    #Res_ID_DITL_Num, d3    ; save the ID Number
        bsr     Save_Text               ; ( both as Str and INT )
        move    #Res_Name_DITL_Num, d3  ; save the Resource Name
        bsr     Save_Text

        lea     Resource_Type_STR, a0
        move.b  (a0), d0                ; get the Length Byte

        cmp     #4, d0
        BEQ.s   @Check_ID               ; length 4 = OK

        move    #103, d3                ; 'ResType must be 4 Chars'
        BRA     Param_Err

@Check_ID

        move    #106, d3                ; assume '0 - 32767' error

        move.l  Resource_ID, d0         ; get the INT form of the ResID
        cmp.l   #32767, d0
        bls     @1                      ; OK so far

        bra     Param_Err               ; whoops - larger than 32767

@1      lea     Resource_ID_Str, a0
        move.b  (a0)+, d0               ; String Length
        cmp.b   #0, d0                  ; length = 0 ?
        bne.s   @2                      ; no

        BRA     Param_Err               ;  no ID entered

@2      move    #107, d3                ; assume 'Resource ID must be 0 ... 9'
                                        ; error
        sub.b   #1, d0                  ; check that each char is digit
        ext.w   d0

@Loop   cmp.b   #'0', (a0)
        bpl.s   @3                      ; OK so far

        BRA     Param_Err

@3      cmp.b   #'9', (a0)+
        dbHI    d0, @Loop
        cmp     #-1, d0                 ; did we exit on an error ?
        bne     Param_Err               ; yes

        move    #0, d4                  ; no error flag

        RTS


Param_Err       ; ResID of Error String in d3
```

```
        move    #-1, d4                 ; errFlag

        pea     Res_DLOG_Storage
        _CloseDialog

        BSR     Restore_Image

        BSR     Stop_Alert
        RTS


Save_Text

        pea     Res_DLOG_Storage
        move    d3, -(sp)               ; Item Number
        pea     ItemType
        pea     ItemHandle
        pea     ItemBox                 ; get the Handle, given
        _GetDItem                       ; the Item Number in D3

        move.l  ItemHandle, -(sp)

        cmp     #Res_Type_DITL_Num, d3
        bne.s   @1
        pea     Resource_Type_STR
        _GetIText

        RTS

@1      cmp     #Res_ID_DITL_Num, d3
        bne.s   @2

        pea     Resource_ID_Str         ; first save the ID as a String
        _GetIText

        lea     Resource_ID_Str, a0     ; then save it as an INTEGER
        move    #StringToNum, -(sp)     ; ( actually a LONG )
        _Pack7
        lea     Resource_ID, a0
        move.l  d0, (a0)

        RTS

@2      pea     Resource_Name
        _GetIText

        RTS


; ----------  CONSTs ( PC-relative )  --------------

Res_DLOG_Storage        dcb.b   170, 0

PutFile_Posit           dc.l    $00340064       ; TopLeft = ( 52, 100 )

SFReply
   good:                dc.b    0
   copy:                dc.b    0
   fType:               dc.l    0
   vRefNum:             dc.w    0
   version:             dc.w    0
   fName:               dcb.b   64, 0

ItemHit                 dc      0
ItemType                dc      0
ItemHandle              dc.l    0
ItemBox                 dcb.l   2, 0

tempString              dcb.b   40, 0

Resource_Type_Str       dcb.b   10, 0   ; allow for overage
Resource_ID_Str         dcb.b   10, 0   ; resID as a String
Resource_Name           dcb.b   20, 0
```

continued

```
Resource_Type           dc.l    0       ; ResType w/out Length Byte
Resource_ID             dc.l    0       ; resID as a LONG ( really INT )


; -----------  VARS ( off A5 )  ------------------

Have_Prior_DITLs        ds.b    1
Default_Vol             ds      1
CurResFile              ds      1


END
```

TRAVERSE.ASM
Contributed by: Howard Katz
"Region Maker," by Howard Katz. January, page 145.

```
; Traverse.ASM          Abstracted from < RgnMaker.ASM >
;                       16 July '86  h. katz

Include MacTraps.D
Include QuickEqu.D      ; gives ScreenBits, rowBytes, bounds for screen

XDEF DoContour

XREF GetFirstPixel, FormingRgn, StartCoords, Trav_Count, MyRowBytes
XREF IsRegion, CreateMenu, WMgrPort, ScratchSTR, PenPoint, WStorage
XREF Stop_Alert

HiBitNum        equ     7
CopyRegionItem  equ     4
Max_Trav_Count  equ     8000            ; an arbitrary upper limit in case
                                        ; of an 'endless' loop

DoContour

        jsr     GetFirstPixel           ; returned -1 = got one
                                        ; anything else = no
        bmi     @0                      ; could we find a 1st Pixel ?
        RTS                             ; guess not


@0      tst.b   FormingRgn(a5)          ; are we doing a Region Def ?
        beq.s   @InitRegs               ; guess not

        tst.b   IsRegion(a5)            ; have we already formed one ?
        beq.s   @1                      ; no

        move.l  a6, -(sp)               ; yes - get rid of it
        _DisposRgn

@1      clr.l   -(sp)
        _NewRgn
        move.l  (sp)+, a6               ; save it for _CloseRgn

        _OpenRgn

        _HidePen

        move.l  StartCoords(a5), -(sp)  ; local Coords of 1st pt w/pixel on
        _MoveTo                         ; do we really have to do this for
                                        ; a Rgn ??  maybe not

@InitRegs

        _HideCursor

        move    #0, Trav_Count(A5)      ; how many On Pixels we've traversed
        move    #0, D7                  ; how many Pixels we've checked
        move    #0, D5                  ; Current Direction = Up ( North )
        movea.l A3, A4                  ; Current Addr = StartPt Addr
        move.w  D3, D4                  ; Current BitNum = StartPt BitNum
        move.l  #64, MyRowBytes(a5)     ; setup rowBytes
        bra     SearchNext              ; GO DO IT


;-------
Traverse
;-------

        ; beginning with the StartPt, do a clockwise traverse around the
        ; contour of the region under consideration, always bearing to
        ; the outside.

        ; A1  Testing Addr      D1  Testing BitNum
        ;                       D2  Testing Direction
        ; A3  Start   Addr      D3  Start   BitNum
        ; A4  Current Addr      D4  Current BitNum
        ;                       D5  Current Direct
        ; A6  Rgn Handle        D6  Current Tries ( 3 Max )
        ;                       D7  Number of tested pixels

        cmp     #Max_Trav_Count, d7     ; an arbitrarily 'high' number
        BHI     No_Loop_Error           ; a 'just-in-case' doublecheck

        cmpa.l  A4, A3                  ; have we completed the loop ?
        bne     SearchNext              ; no
        cmp.b   D4, D3                  ; back at the same BitNum ?
        bne     SearchNext              ; no

        ; We've returned to our Starting Point

        _ShowCursor             ; ( blanked when we pointed to StartPt )

        tst.b   FormingRgn(a5)
        beq.s   @9              ; no

        move.l  StartCoords(a5), -(sp)  ; close the loop
        _MoveTo
        move.l  a6, -(sp)
        _CloseRgn

        st      IsRegion(a5)            ; We can now do a Region 'Copy'

        move.l  CreateMenu(A5), -(sp)
        move    #CopyRegionItem, -(sp)
        _EnableItem

@9      _PenNormal
        _ShowPen

        move    #2, -(sp)
        _SysBeep

        move.l  WMgrPort, -(sp)         ; so we draw in the MenuBar
        _SetPort

        move.l  (a5), a0                ; thePort
        pea     bounds+ScreenBits(a0)   ; for ClipRect
        _ClipRect

        pea     MenuRect
        _EraseRect

        lea     MenuRect, a0
        move.l  Left(a0), d0            ; LeftBottom
        sub     #3, d0                  ; Bottom - 3 ( move pen up a bit )
        swap    d0                      ; BottomLeft
        move.l  d0, -(sp)

        _MoveTo
        move    Trav_Count(A5), d0
        ext.l   d0
        lea     ScratchSTR, a0
        move    #0, -(sp)               ; Num to String
        _Pack7
        pea     ScratchSTR
        _DrawString
```

continued

```
        pea     '/'
        _DrawString

        move    d7, d0
        ext.l   d0
        lea     ScratchSTR, a0
        move    #0, -(sp)
        _Pack7
        pea     ScratchSTR
        _DrawString

        pea     ' pixels'
        _DrawString

        pea     PenPoint
        _GetPen                         ; save for Region Size display

        pea     WStorage                ; reset Port to our Window
        _SetPort

;       ===
        RTS                             ; traverse ALL DONE
;       ===

;---------
SearchNext
;---------

        move    #2, D6                  ; allow up to 3 tries / else ABORT

Try_S_Minus_1_Neighbor

        move.b  D5, D2                  ; try the (S-1)-Neighbor
        sub.b   #1, D2                  ; Direction to Try
        and     #7, D2

        jsr     Test_Bit                ; locate and test apt pixel
        beq.s   @Try_S_Neighbor         ; the pixel was off

        sub.b   #2, D5                  ; rotate search direction 3 CCW
        and     #7, D5
        bra.s   @PixelWasOn             ; and search for next pixel in contour

@Try_S_Neighbor

        move    D5, D2                  ; try the S-Neighbor
        jsr     Test_Bit                ; locate and test apt pixel
        bne.s   @PixelWasOn             ; ON - continue in same direction

@Try_S_Plus_1_Neighbor

        move    D5, D2                  ; try the (S+1)-Neighbor
        add.b   #1, D2
        and     #7, D2
        jsr     Test_Bit                ; update BitNumber and (a3) to test
        beq.s   Rotate_3_CW             ; not on

@PixelWasOn

        movea.l A1, A4                  ; pixel GOOD - update address
        move    D1, D4                  ; update bit number in current byte

        move    Trav_Count(A5), D0
        add     #1, D0                  ; Inc Count of Pixels Traversed
        move    D0, Trav_Count(A5)

; if we're forming a Rgn, set up the appropriate coords ( from D5)
; for a _Move command

        tst.b   FormingRgn(a5)  ; are we building a region ?
        beq     Traverse        ; no - go find next point

        move    D2, d0          ; the Direction we successfully moved in
        add     d0, d0
        add     d0, d0          ; X 4 ( each entry is 4 bytes )
        lea     Line_Table, a0
        move.l  0(a0, d0), d0   ; get the appropriate pair of coords
        move.l  d0, -(sp)
        _Line                   ; the whole object of the exercise

        bra     Traverse        ; and look for next point

Line_Table

; dir           d_Y / d_X

@0      dc.l    $FFFF0000
@1      dc.l    $FFFF0001
@2      dc.l    $00000001
@3      dc.l    $00010001
@4      dc.l    $00010000
@5      dc.l    $0001FFFF
@6      dc.l    $0000FFFF
@7      dc.l    $FFFFFFFF


Rotate_3_CW

        add.b   #3, D5                  ; rotate search direction 2 Clockwise
        and     #7, D5

        DBRA    D6, Try_S_Minus_1_Neighbor

; if we're here, we've exceeded 3 TRIES of the above loop
; or we're stuck in some other sort of endless loop


No_Loop_Error

        _ShowCursor

        move.l  StartCoords(a5), -(sp)  ; close the loop
        _MoveTo

        tst.b   FormingRgn(a5)
        beq.s   @1

        move.l  a6, -(sp)
        _CloseRgn

@1      _ShowPen

        move    #101, d3        ; ResID of upcoming Str message
        BSR     Stop_Alert      ; 'Couldn't find a Closed Loop'

        RTS                     ; to MainLoop


Test_Bit

        ; depending on the Direction we're Testing ( D2 ), change
        ; the BitNumber ( D1 ) and where A1 is pointing if necessary.

        ;  D1  Testing BitNum   A1  Testing Addr
        ;  D2  Testing Direct

        movea.l A4, A1                  ; setup Test Address
        move    D4, D1                  ; setup Test BitNum

        lea     JTable, a0
        move    D2, d0
        asl     #1, d0
        adda.w  0(a0, d0), a0
        jmp     (a0)


RetAddr

        add     #1, d7                  ; one more tested Pixel
        tst.b   FormingRgn(A5)
        beq.s   @9
        btst.b  D1, (A1)                ; Test the Bit
```

continued

```
        bra.s   @rts
@9      bclr.b  D1, (A1)                ; Test then Clear the Bit onscreen

@rts    RTS


JTable:

@0      dc      MoveVertical    - JTable        ; 0  = N
@1      dc      MoveRight       - JTable        ; 1  = NE
@2      dc      MoveRight       - JTable        ; 2  = E
@3      dc      MoveRight       - JTable        ; 3  = SE
@4      dc      MoveVertical    - JTable        ; 4  = S
@5      dc      MoveLeft        - JTable        ; 5  = SW
@6      dc      MoveLeft        - JTable        ; 6  = W
@7      dc      MoveLeft        - JTable        ; 7  = NW


MoveVertical
        cmp     #0, D2          ; moving Up ?
        beq     MoveUp          ; yes

MoveDown
        adda.l  MyRowBytes(a5), A1 ; moving DOWN - A1 points to byte BELOW

        BRA.s   RetAddr

MoveUp
        suba.l  MyRowBytes(a5), A1      ; point to byte ABOVE

        BRA.s   RetAddr

MoveRight
        sub     #1, D1          ; point to next pixel to right
        cmp     #0, D1          ; have we passed bitNumber 0 ?
        bpl     @CkRight        ; no
        move    #HiBitNum, D1   ; reset bitNumber to 8
        adda.l  #1, A1          ; point to next higher Byte to check
@CkRight
        cmp     #1, D2          ; moving NE ?
        beq     MoveUp          ; yes
        cmp     #2, D2          ; moving E ?
        beq     RetAddr         ; no
        bra.s   MoveDown        ; must be SE


MoveLeft
        add     #1, D1          ; BitNum points to next Leftmost pixel
        cmp     #HiBitNum, D1   ; greater than 7 ?
        bls     @CkLeft         ; no / else
        move    #0, D1          ; zero BitNum = Rightmost pixel in next byte
        suba.l  #1, A1          ; point to byte to Left

@CkLeft
        cmp     #7, D2          ; moving NW ?
        beq     MoveUp          ; yes
        cmp     #6, D2          ; moving W ?
        beq     RetAddr
        bra.s   MoveDown        ; must be SW


MenuRect        dc      0, 280, 18, 505

        END
```

LIST.1
Contributed by: Jon C. Snader
Programming Project: "Look it Up Faster with Hashing," by Jon C. Snader.
January, page 128.

```
50 '
55 '
60 'Subroutine to hash a four character string
65 '
70 'Enter with the string to be hashed in NA$
75 '
80 'On exit the hash value is in H
85 '
90 '
95 '
100 H = ( CVI(MID$(NA$,1,2)) XOR CVI(MID$(NA$,3,2)) ) MOD 61
110 RETURN
```

LIST.2
Contributed by: Jon C. Snader
Programming Project: "Look it Up Faster with Hashing," by Jon C. Snader.
January, page 128.

```
      Function h( KEY: string4 ): integer;
         Type
            KEY_types = (char_KEY, integer_KEY);
            KEY_overlay = record
               case KEY_types of
                  char_KEY:    ( KEY_in_characters: string4 );
                  integer_KEY: ( dummy: byte; {takes up room for string size}
                                 integer_KEY_1: integer; {first 2 bytes of
KEY}
                                 integer_KEY_2: integer; {last 2 bytes of KEY}
                                );
            end;

         Var
            KEY_record: KEY_overlay;
         begin {hash}
         with KEY_record do
            begin
               KEY_in_characters := '    '; {clean out in case KEY < 4 chars}
               KEY_in_characters := KEY;
               h := ( integer_KEY_1 xor integer_KEY_2 ) mod
number_TAB_entries;
            end;
         end; {hash}
```

LIST.3
Contributed by: Jon C. Snader
Programming Project: "Look it Up Faster with Hashing," by Jon C. Snader.
January, page 128.

```
C       **********************************************************
C       *                                                        *
C       * A SUBROUTINE TO CALCULATE A HASH VALUE BETWEEN         *
C       * 0 AND 60                                               *
C       *                                                        *
C       *                                                        *
C       * INPUT:  KEY - FOUR BYTES OF CHARACTER DATA TO BE HASHED *
C       *                                                        *
C       *                                                        *
C       * OUTPUT: INDEX - AN INTEGER VALUE BETWEEN 0 AND 60      *
C       *                                                        *
C       **********************************************************
C
```

continued

```
      SUBROUTINE HASH(KEY,INDEX)
      CHARACTER KEY*4,WKEY*4
      INTEGER*2 INDEX,IKEY(2),EOR
      EQUIVALENCE (WKEY,IKEY)
      WKEY=KEY
      IKEY(1)=EOR(IKEY(1),IKEY(2))
      INDEX=MOD(IKEY(1),61)
      RETURN
      END
```

LIST.4
Contributed by: Jon C. Snader
Programming Project: "Look it Up Faster with Hashing," by Jon C. Snader.
January, page 128.

```
;---------------------------------------------
;HASH--Procedure to hash a four byte string
;
;       Input:  AX := first two bytes of string
;               BX := second two bytes of string
;
;       Output: AL := hash value (0-60)
;
;       Registers destroyed:    AX,BX
;---------------------------------------------
;
TAB_sz  equ     61              ;define table size
hash    proc    near
        push    dx              ;save DX
        xor     ax,bx           ;combine into 16 bits
        xor     dx,dx           ;clear DX-dividend in DX AX
        mov     bx,TAB_sz       ;table size to BX
        div     bx              ;divide-remainder is in DX
        mov     ax,dx           ;remainder to AX
        pop     dx              ;restore DX
        ret
hash    endp
```

LIST.5
Contributed by: Jon C. Snader
Programming Project: "Look it Up Faster with Hashing," by Jon C. Snader.
January, page 128.

```
100 'Routine to do a table look-up using chained hashing
110 '
120 'TB = table of names to be entered/looked up.
130 'CH = table of chain pointers
140 'IX = index to entry of TB where the name was entered or found
150 'OV = pointer to the last entry used in the overflow table
160 'FD = flag reporting result of search: 0=not found, 1=found
170 'K$ = holds the current KEY being searched for
180 'MT = maximum total table size (primary and secondary)
190 '
200 FD = 0  'initialize result of search to "not found"
210 GOSUB 1000 'go hash the key in K$; the result is returned in IX
220 '
230 'examine first entry with correct hash value
240 '
250 IF TB(IX) = "" THEN TB(IX) = K$: RETURN 'it's empty - enter KEY and return
260 IF TB(IX) = K$ THEN FD = 1: RETURN  'found it - say so and return
270 '
280 'the first entry had some name other than KEY in it - step down the chain
290 '
300 IF CH(IX) <> 0 THEN IX = CH(IX): GOTO 260 'step down the chain
310 '
320 'We found the end of the chain,  so enter the key and return with FD = 0
330 '
340 OV = OV + 1  'advance to next empty overflow entry
350 IF OV > MT THEN GOTO 2000 'goto the error routine and never return
360 TB(OV) = K$   'enter KEY
```

```
370 CH(IX) = OV   'and add the new entry to the end of the chain
380 IX = OV       'set IX to tell the caller where we entered it
390 '
400 RETURN
```

LIST.6
Contributed by: Jon C. Snader
Programming Project: "Look it Up Faster with Hashing," by Jon C. Snader.
January, page 128.

```
Program Search_With_Chaining;

   Const
      max_TAB_entry = 60;  {last TAB entry number}
      number_TAB_entries = 61; {the number of entries in TAB}

   Type
      tab_pointer = ^tab_entry;  {define a pointer to tab_entry (below)}
      string4 = string[4];
      tab_entry = record        {define an entry of TAB}
         KEY_field: string4;   {holds KEY for this entry}
         CHAIN: tab_pointer;    {pointer to next entry with same hash value}
      end;

   Var
      found: boolean;  {set true by Search if KEY is found}
      index: tab_pointer; {pointer to the current TAB entry being examined}
      KEY: string4;  {name to be found or entered}
      i: integer;  {for FOR loop use}
      node: array[ 0 .. max_TAB_entry ] of tab_pointer; {heads for each chain}

   Procedure Search( KEY: string4 );

      Function h( KEY: string4 ): integer;
         Type
            KEY_types = (char_KEY, integer_KEY);
            KEY_overlay = record
               case KEY_types of
                  char_KEY:     ( KEY_in_characters: string4 );
                  integer_KEY: ( dummy: byte; {takes up room for string size}
                                  integer_KEY_1: integer; {first 2 bytes of
KEY}
                                  integer_KEY_2: integer; {last 2 bytes of KEY}
                                 );
            end;

         Var
            KEY_record: KEY_overlay;
         begin {hash}
         with KEY_record do
            begin
               KEY_in_characters := '    '; {clean out in case KEY < 4 chars}
               KEY_in_characters := KEY;
               h := ( integer_KEY_1 xor integer_KEY_2 ) mod
number_TAB_entries;
            end;
         end; {hash}
      Var
         hash: integer;  {holds the hash value the current KEY}
         last_index: tab_pointer; {points to the last entry examined}

      Begin {Search}
         found := false;
         hash := h( KEY ); {go hash KEY}
         index := node[ hash ];
         if index = nil then {this is the first KEY with this hash value}
            begin
               new( index );  {create an entry for it}
               node[ hash ] :=  index; {and set node to point to it}
               index^.CHAIN := nil; {mark this entry as the end of the chain}
               index^.KEY_field :=  KEY; {enter KEY into TAB entry}
            end
```

continued

```
        else {there are entries with this hash value - search them}
           begin
              while ( index <> nil ) and not found  do
                  begin
                     if index^.KEY_field = KEY then {found it}
                        found := true
                     else {point to next entry with this hash value}
                        begin
                           last_index  := index; {point to the LAST entry}
                           index := index^.CHAIN;
                        end;
                  end;
                if not found then {create a new entry}
                  begin
                     new( last_index^.chain );
                     index := last_index^.chain; {and point to it with index}
                     index^.CHAIN := nil; {mark this entry as end of chain}
                     index^.KEY_field := KEY; {enter KEY into TAB entry}
                  end;
         end;
   end; {Search}
Begin {Search_With_Chaining}

   for i:=0 to max_TAB_entry do node[i]:=nil; {set nodes to point nowhere}


           {User Code Goes Here}


End. {Search_With_Chaining}
```

LIST.7
Contributed by: Jon C. Snader
Programming Project: "Look it Up Faster with Hashing," by Jon C. Snader.
January, page 128.

```
Program Search_With_Double_Hashing;

   Const
      max_TAB_entry = 60;       {last TAB entry}
      number_TAB_entries = 61;  {number of entries in TAB}
      empty = '    ';           {what an empty entry looks like}
      p_prime = 59;             {first twin prime-used to calculate increment}
      p = 61;                   {second twin prime-used to hash KEY}

   Type
      string4 = string[4];

   Var
      found: boolean;           {set true by search if KEY is found}
      index: integer;           {pointer to the TAB entry being examined}
      KEY: string4;             {name to found or entered}
      i: integer;               {for FOR loop use}
      n: integer;               {number of entries currently in TAB}
      TAB: array[ 0 .. max_TAB_entry ] of string4;

   Procedure Search( KEY: string4 );

      Function h( KEY: string4; modulus: integer ): integer;

         Type
            KEY_types = (char_KEY, integer_KEY);
            KEY_overlay = record
               case  KEY_types of
                  char_KEY:      ( KEY_in_characters: string4);
                  integer_KEY:   ( dummy: byte; {takes up room for string
size}
                                   integer_KEY_1: integer; {first 2 bytes}
                                   integer_KEY_2: integer; {last 2 bytes} );
            end;

         Var
            KEY_record: KEY_overlay;
      begin {h}
        with KEY_record do
          begin
            KEY_in_characters := '    '; {in case KEY < 4 chars}
            KEY_in_characters := KEY;
            h := ( integer_KEY_1 xor integer_KEY_2 ) mod modulus;
          end;
      end; {h}
    Procedure add_KEY_to_TAB;
      begin {add_KEY_to_TAB}
        n := n + 1;  {one more entry in TAB}
        if n > max_TAB_entry then {table is full}
          begin
            writeln('     ***Fatal Error***');
            writeln('Table overflow in table TAB');
            writeln('      program aborted');
            halt; {stop with a fatal error}
          end
        else {there's still room, so add another entry}
          TAB[ index ] := KEY;
      end; {add_KEY_to_TAB}

    Var
      j: integer;                 {increment for current KEY}

    begin {search}
      found := false;
      index := h( KEY, p );  {go hash KEY}
      if TAB[ index ] = KEY then {found it}
        found := true
      else {we have to do some more looking}
        begin
          if TAB[ index ] = empty then {it's not there - enter it}
            add_KEY_to_TAB
          else
            begin
              j := h( KEY, p_prime ) + 1;  {calculate the increment}
              repeat
                index := index + j;  {step index to next entry}
                if index > max_TAB_entry then {off the end of TAB}
                  index := index - number_TAB_entries; {make
circular}
                if TAB[ index ] = KEY then {we found it}
                  found := true;   {so say so}
              until ( TAB[ index ] = empty )  or found;
              if not found then {we need to enter KEY}
                add_KEY_to_TAB;  {so do so}
            end;
        end;
    end; {search}

  Begin {Search_With_Double_Hashing}
    n := 0;  {no entries in TAB yet}
    for i := 0 to max_TAB_entry do TAB[ i ] := empty; {all entries
available}



    {user code goes here}



  End. {Search_With_Double_Hashing}
```

February

IFP.TXT
Contributed by: Arch D. Robison
"Illinois Functional Programming: A Tutorial" by Arch D. Robison, page 114.

IFP Reference 1

1. Built-in Functions

This section is a reference guide to the built-in functions in IFP. The following sets (types) are used in the definitions of functions:

A	atoms
B	boolean values
O	objects
R	real numbers
Z	integers
S	strings
T*	sequences with element type T
T+	non-empty sequences with element type T
Tn	sequences of length n with element type T

A function returns ''?'' if the argument is not in its domain. The notation xn denotes the nth element of a sequence X.

For example, the domain of the addition function is [X,Y] in [R,R]. That is addition takes a pair of real numbers as its argument. We could also write this as [X,Y] in R2, since a pair is a sequence of length two.

December 5, 1985
IFP Reference 2

1.1. Structural Functions (/sys)

Structural functions are assemble, reorganize, and select data. The primitive structural functions are listed below:

Name	Domain	Definition
apndl	[X,Y] in [O,On]	<X, y1 , y2 , ...yn>
apndr	[X,Y] in [Om,O]	<x1, x2, ... xm, Y>
cat	X in O**	catenate sequences
distl	[X,Y] in [O,On]	<<X,y1> <X,y2> ... <X,yn>>
distr	[X,Y] in [Om,O]	<<x1,Y> <x2,Y> ... <xm,Y>>
dropl	[X,K] in [On, 0≤Z≤n]	drop K elements from left end of X
dropr	[X,K] in [On, 0≤Z≤n]	drop K elements from right end of X
iota	n in Z≥0	<1,2,...n>
length	X in On	number of elements in X
pick	[X,K] in [On, 0<Z≤n]	Kth element of X
repeat	[X,K] in [O,0≤Z]	sequence <X,X...X> of length K
reverse	X in On	reversal of X
takel	[X,K] in [On, 0≤Z≤n]	take K elements from left end of X

continued

taker	[X,K] in [On, 0≤Z≤n]	take K elements from right end of X
tl	X in O+	(tail) drop first element of X
tlr	X in O+	(right tail) drop last element of X
trans	X is matrix	transpose X

1.2. Arithmetic (/math/arith)

Most IFP arithmetic functions are found here. Below is a table of the existing functions. Some function's domain may be further restricted due to range limitations.

Name	Domain	Definition
+	[X,Y] in [R,R]	X+Y
-	...	X-Y
*	...	XxY
%	[X,Y] in [R,R≠0]	X/Y
add1	X in R	X+1
arcsin	X in R, -1≤X≤1	arcsine X
arccos	X in R, -1≤X≤1	arccosine X
arctan	X in R	arctangent X
cos	X in R	cosine X
div	[X,Y] in [R,R≠0]	floor (X/Y)
exp	X in R	e to the Xth power
ln	X in R>0	natural logarithm of X
max	[X,Y] in [R,R]	maximum of X and Y
min	[X,Y] in [R,R]	minimum of X and Y
minus	X in R	-X
mod	[X,Y] in [R,R]	X modulo Y
power	[X,Y] in [R≥0,R]	X to Yth power
sin	X in R	sine X
sqrt	X in R>0	square root of X
sub1	X in R	X-1
sum	X in R*	summation of X
tan	X in R	tangent of X

1.3. Logic (/math/logic)

Most IFP primitive functions returning boolean values are found here. Below is a table of the existing functions:

Name	Domain	Definition
=	[X,Y] in [O,O]	X=Y
~=	...	X≠Y
<	[X,Y] in [R,R] u [S,S]	X<Y
<=	...	X≤Y
>=	...	X≥Y
>	...	X>Y
~	X in B	not X
and	[X,Y] in [B,B]	X AND Y
all	X in B*	all elements of X are true
any	X in B*	at least one element of X is true
atom	X in O	X is an atom
boolean	X in O	X is boolean
false	X in O	X is #f
imply	[X,Y] in [B,B]	~X OR Y
longer	[X,Y] in [Om,On]	m>n
member	[X,Y] in [O*,O]	Y is an element of X
numeric	X in O	X is a number
null	X in O*	X = <>
odd	X in Z	X is odd
or	[X,Y] in [B,B]	X OR Y
pair	X in O	X is a pair
shorter	[X,Y] in [Om, On]	m<n
xor	[X,Y] in [B,B]	X≠Y

String inequalities are defined from the lexigraphical (dictionary) ordering.

1.4. String Functions (/sys)

The string functions are:

Name	Domain	Definition
explode	X in S	sequence of characters in X
implode	X in S*	string made by catenating strings in X
patom	X in A	string representation of X

continued

1.5. Miscellaneous Functions (/sys)

The miscellaneous functions are listed below. Each function description is preceded by a title line of the form:

```
function                      domain                     definition
```

```
apply                       [X,F] in [O,S*]              apply F to X
```

F is a sequence of strings representing a path to a defined function. The result is the function referenced by F applied to X. Example:

```
        <<3 4> <math arith "+">> : apply -> 7
```

```
assoc                     [X,Y] in [(O+)*,O]      associative lookup
```

X is an association sequence, which is a sequence of non-empty subsequences. The first element of each subsequence is the key of the subsequence. The result of assoc is the first subsequence of X with a key equal to Y. If no matching key is found, f is returned. The key may be any type of object. Examples:

```
        <<<a b c> <w x y z> <i j>> w> -> <w x y z>
        <<<a b c> <w x y z> <i j>> U> -> f
```

December 5, 1985
IFP Reference 6

```
def                             X in S+                   definition
```

The definition function returns the object representation of its argument. The representation of a function is a sequence of strings denoting its absolute path. The representation of a PFO is a sequence. The first element of the sequence is a path to the PFO. The remaining elements of the sequence are parameters of the functional form. Suppose, for example, we define the inner product function:

```
        DEF Inner AS trans | EACH * END | INSERT + END
```

and ''Inner'' is defined with a module with path ''/math/linear''. Then ''<math linear Inner> : def'' will result in:

```
          <
                <sys compose>
                <sys trans>
                <<sys each> <math arith *>>
                <<sys insertr> <math arith +>>
          >
```

Currently, the representations of PFO are:

```
#c                        <<sys constant> #c>
#?                        <<sys constant>>
n                         <<sys selectl> n>
nr                        <<sys selectr> n>
f1 | f2 | ... fn          <<sys compose>, f1 , f2 , ... fn>
[f1 , f2 , ... fn ]       <<sys construct>, f1 , f2 , ... fn>
^c                        <<sys fetch> c>
EACH f END                <<sys each> f>
FILTER p END              <<sys filter> p>
INSERT f END              <<sys insertr> f>
IF p THEN g ELSE h END    <<sys if> p g h>
WHILE p DO f END          <<sys while> p f>
```

ELSIF clauses are always expanded into equivalent nested IF-THEN-ELSE constructions. Note the special case for #?, the representation <<sys constant> ?> would be useless due to the bottom-preserving property.

```
id                         X in O                         identity
```

The identity function returns its argument. It is useful as a place holder in PFO. For example, the ''square'' function can be written as:

```
DEF Square AS [id,id] | *;
```

December 5, 1985
IFP Reference 7

2. Program Forming Operations

Program forming operations combine functions and objects to create new functions.

2.1. Constant

Constant functions always return the same result when applied to any value which is not ''?''. Constant functions are written as:

```
#c
```

where c is the constant value to be returned. A constant function applied to ''?'' results in ''?''. Note that the function ''#?'' always returns '?'. Examples:

```
923 : #<cat in hat> -> <cat in hat>
<a b c d e f> : #427 -> 427
? : #<q w er t y> -> ?
5 : #? -> ?
```

2.2. Selection

Selector functions return the nth element of a sequence and are written as n, where n is a positive integer. Note the distinction between #5, which returns the value 5, and 5, which returns the fifth element of its argument. There are also a corresponding set of select-from-right functions, written as nr. These select the nth element of a sequence, counting from the right. All selectors return ''?'' if the argument has no nth element or is not a sequence. Below are some examples of applying selector functions:

```
<a b c d e> : 1 -> a
<a b c d e> : 2 -> b
<apple banana cherry> : 1r -> cherry
<apple banana cherry> : 4 -> ?
```

December 5, 1985
IFP Reference 8

```
hello : 1 -> ?
```

2.3. Composition

The function composition of two functions is written as:

```
f | g
```

Applying the result function is the same as applying f and then g. E.g.: Function composition is defined by the equality:

```
x : (f | g) ≡ (x : f) : g
```

Since function composition is associative, the composition of more than two functions does not require parentheses. The composition of f1,f2,...fn is written:

```
f1 | f2 | ...fn
```

Composition syntax is identical to UNIX's pipe notation for a reason: function composition is isomorphic to a pipe between processes without side effects.

continued

2.4. Construction

The construction of functions is written as bracketed list of the functions. For example, the construction of functions fi is written:

```
                    [f1,f2,...fn]
```

Function construction is defined by the equality:

```
          x : [f1,f2,...fn] ≡ <x:f1,x:f2,...x:fn>
```

2.5. Apply to Each

The EACH functional form applys a function to each element of a sequence. It is written as

```
    EACH f END
```

It is defined by the equality:

```
    <x1,x2,...xn> : EACH f END ≡ <x1:f,x2:f,...xn:f>
```

2.6. If-Then-Else

The IF functional form allows conditional function application. It is written as

```
    IF p THEN g ELSE h END
```

and is defined by the equality:

```
                                   | x:g  if p=#t
                                   |
    x : IF p THEN g ELSE h END ≡   | x:h  if p=#f
                                   | ?   otherwise
                                   |
```

The level of nesting of conditional forms may be reduced by using ELSIF clauses:

```
    IF p1 THEN f1 ELSIF p2 THEN f2 ELSIF ... ELSE g END
```

2.7. Filter

The FILTER functional form filters through elements of a sequence satisfying a predicate. It is written as:

```
    FILTER p END
```

where p is the predicate. It is defined by the functional equality:

```
    FILTER p END ≡ EACH IF p THEN [id] ELSE [ ] END END | cat
```

For example, if you wish to find all numeric elements in a sequence, you could write:

```
    FILTER numeric END
```

The FILTER functional form is an IFP extension to Backus' FP.

2.8. Right Insert

The INSERT functional form is defined by the recursion:

```
    INSERT f END ≡ IF tl|null THEN 1 ELSE [1,tl | INSERT f END] | f END
```

Typically it is used for crunching a sequence down. For example,

```
        INSERT + END
```

returns the sum of a sequence.

Unlike Backus' FP, functions formed with INSERT are always undefined for empty sequences. The reason is that it is impractical for the interpreter to know the identity element of user-defined functions. The number of cases where the interpreter could know the identity element are so few that you might as well define special functions for those cases, e.g:

```
        DEF sum AS IF null THEN #0 ELSE INSERT + END END;
```

Alternatively, you can append the identity element to the end of the sequence before inserting, e.g.:

```
        DEF sum AS [id,#0] | apndr | INSERT + END;
```

Currently there is no ''left insert'' form.d The left insertion of f can be written as:

```
        reverse | INSERT reverse|f END
```

December 5, 1985
IFP Reference 11

2.9. While

The WHILE functional form allows indefinite composition. It is written as:

```
        WHILE p DO f END;
```

and is defined by the recursive functional equality:

```
        WHILE p DO f END ≡ IF p THEN f | WHILE p DO f END ELSE id END
```

2.10. Fetch

The fetch functional form allows easy access to association sequences (see function /sys/assoc for a description of association sequences.) A fetch is written as ^c, where c is an object. The fetch form is defined by the functional equality:

```
        ^c ≡ IF EACH pair END | all THEN [id,#c]|assoc|2
             ELSE #?
             END;
```

Note that the input is restricted to a sequence of pairs.

3. Comments

Comments are delimited by matching pairs of ''(*'' and ''*)''. Comments may be inserted anywhere not adjacent to a token. For example:

```
        DEF foo AS bar; (* This is a comment.  DEF foo AS bar is not a comment *)
```

4. Syntax Summary

Below is an EBNF grammar for IFP:

December 5, 1985
IFP Reference 12

```
|Def ->             'DEF String 'AS' Comp ';'                          |
|Comp ->             Simple { '|' Simple }                             |
|Simple ->           Conditional | Constant | Construction | Each | Filter ||
```

continued

```
                    Insert   | Path | While | Fetch | Debug
Conditional ->      'IF' Comp 'THEN' Comp { 'ELSIF' Comp 'THEN' Comp }
                    'ELSE' Comp 'END'
While ->            'WHILE' Comp 'DO' Comp 'end'
Insert ->           'INSERT' Comp 'END'
Each ->             'EACH' Comp 'END'
Filter ->           'FILTER' Comp 'END'
Fetch ->            '^' String
Constant ->         '#' Object
Debug ->            '@' Object
Construction ->     '[' [Comp {',' Comp}] ']'
Path ->             ['/'] String {'/' String}
Object ->           Bottom | Atom | '<' [Atom {','Atom}] '>'
Bottom ->           '?'
Atom ->             Number | String | Boolean
Boolean ->          't' | 'f'
```

Strings may be in single or double quotes. The strings ''t'' and ''f'' must be quoted to distinguish them from boolean atoms. Strings of digits must also be quoted to distinguish them from numeric atoms.

5. Running IFP with MS-DOS

5.1. Prerequisite Hardware

The MS-DOS version needs at least a 256K system. Extra memory for a RAM-disk is convenient but not necessary.

5.2. Prerequisite Software

There are three programs you will need: the IFP interpreter (IFP.EXE), a text editor, and a directory lister. You must supply the text editor and directory lister. (The ''PC-Write'' editor works with IFP under DOS 2.0 and 3.0; ''edlin'' only works under DOS 3.0; I haven't tried any others). All three of these programs must reside on a different disk drive than your IFP functions. If you have enough memory, it is advantageous to put these on a RAM-disk. The IFP function files should be kept on a floppy or hard disk, just in case your machine crashes.

5.3. Running IFP

Before invoking IFP, two environment variables should be set. The ''EDITOR'' variable should be set to the name of your favorite editor. The default editor is ''c:ed.exe''. The ''FPDIR'' variable should be set to the name of your favorite directory listing program. Normally these variables should be set by the autoexec.bat file. Below is an example autoexec.bat file:

```
set EDITOR = A:edlin.com
set IFPDIR = A:sd2.com
```

5.4. Starting IFP

To start an IFP session, change your current working directory to a directory on the IFP functions disk. Then execute the ''ifp.exe'' program. Your current working directory becomes your current working IFP module. (There is no way to change your current working directory from within IFP. To change it, leave the interpreter and change it within DOS.) When IFP is ready, it will respond with the prompt ''ifp> ''. To end the IFP session, enter the command ''exit''. All function definitions are kept in disk files, so you can't lose anything when you exit or the computer crashes.

To edit an IFP definition file, type the command:

```
ed name
```

where *name* is the name of the function to be edited. (Since all IFP reserved words are upper case, it is a good practice to use lower or mixed case for function names.) The function may be one local to the current working module, or one that is imported into the current working module. If the function name is neither defined locally nor imported, then it is assumed to be a new local function. The function definition file must be of the form:

```
        DEF name AS f;
```

Definitions are in free format, line breaks are treated as spaces. Matching pairs of ''(*'' and ''*)'' delimit comments as in Pascal. Note: Do not switch to another file from within the editor. Always exit the editor to return to the IFP command interpreter first and then edit the next file. Otherwise interpreter won't know that its internal copy of a function is invalid.

To apply an IFP function, type the command:

```
        show object : function;
```

The interpreter evaluates the result of applying the function to the object. The result is then pretty-printed at the terminal. Listing 1 shows a sample session.

To list your functions, type the command:

```
        dir
```

The directory listing program specified by IFPDIR will be invoked. *Note*: my directory lister won't work unless I type a trailing slash, i.e. ''dir/". I have not tried any other directory listing programs.

To delete a function, type the command:

```
        del f
```

The function definition file (along with the memory copy) will be deleted. Wildcards are not permitted in the function name.

Warning: do not try to delete files with extensions (e.g. ''.bak'') from within IFP, since file names are truncated to 8 characters, IFP may delete the wrong file.

5.5. *Tracing Functions*

Currently, IFP has simple program trace mechanism. To trace a function, respond to the IFP prompt with:

```
        trace on f ,f ,...f ;
                  1  2     n
```

where the f's are functions to be traced. Whenever a traced function is invoked, its argument and result are shown. Also, the argument and result of all called functions are shown. To stop tracing functions, respond to the IFP prompt with:

```
        trace off f ,f ,...f ;
                   1  2     n
```

When tracing, the interpreter ellipses are used to abbreviate functions. You can set the depth at which ellipses occur with the *depth* command:

```
        depth n
```

where n is a non-negative integer. The default depth is two.

continued

There is also a functional form for creating trace functions. Its form is

```
        @string
```

The function formed always returns its argument unchanged, and it prints ''string: '' followed by its argument. For example,

```
        <1 3 5> : EACH @banana END
```

will print the messages:

December 5, 1985
IFP Reference 16

```
        banana: 1
        banana: 2
        banana: 3
```

This tracing functional form is for debugging only, since it creates a side effect (the message!), it is not truly functional.

Program execution can be aborted at any time by pressing control-C. A trace of where the function was will be shown. Pressing control-C again will abort the trace.

December 5, 1985

POLY.BAS
Contributed by Rene Stolk and George Ettershank
"Calculating the Area of an Irregular Shape," by Rene Stolk and George Ettershank, page 135.

```
10 DIM X(50),Y(50)
20 READ N
30 FOR K=1 TO N
40 READ X(K),Y(K)
50 NEXT K
60 X(0)=X(N)
70 Y(0)=Y(N)
80 AREA=0
90 FOR K=0 TO N-1
100 AREA=AREA + X(K) * Y(K+1) - X(K+1) * Y(K)
110 NEXT K
120 AREA = .5 * ABS(AREA)
130 PRINT "Enclosed area is "; AREA
140 END
150 DATA 4
160 DATA 4,3,4,1,1,4,3,4
```

LISTING1
Contributed by Robert J. Sciamanda
"Another Approach to Data Compression," by Robert J. Sciamanda, page 137.

```
10 OPEN "O",#1,"DATA" :REM Make a test data file by taking
20 PRINT#1,50    :REM 51 samples from a Gaussian curve
30 D=.2          :REM centered at i=25.
40 FOR I=0 TO 50
50 A=10*EXP(-(5-D*I)^2)
60 PRINT#1,A
70 NEXT I
80 CLOSE
```

LISTING2
Contributed by Robert J. Sciamanda
"Another Approach to Data Compression," by Robert J. Sciamanda, page 137.

```
10 OPEN "I",#1,"DATA"
20 INPUT#1,N    :REM Get data count.
30 DIM A(N)
40 FOR I=0 TO N :REM Get original data set.
50 INPUT#1,A(I)
60 NEXT I
70 CLOSE
80 INPUT "Enter desired accuracy ";E
90 FOR L=2 TO INT(N/2-.5)
100 W=3.141593/L
110 FOR I=1 TO N          :REM Reconstruct missing values.
120 IF I MOD L=0 THEN 190 :REM Branch at sampled values.
130 G=0
140 FOR J=0 TO N STEP L :REM The Nyquist sum.
150 M=W*(I-J)
160 G=G+A(J)*SIN(M)/M
170 NEXT J
180 IF ABS(G-A(I))>E THEN 210 :REM Sum done; test accuracy.
190 NEXT I       :REM If ok, reconstruct next value.
200 NEXT L       :REM Increment sampling interval.
210 L=L-1        :REM Highest successful sampling interval.
220 IF L>1 THEN 260      :REM L=1 means no compression possible.
230 PRINT "For an accuracy of +/-";E;"all of this data
    must be kept."
240 PRINT "No compressed data file (CDATA) will be generated."
250 END :REM Exit.
260 OPEN "o",#1,"CDATA" :REM Create compressed data file.
270 PRINT#1,N,L :REM Write data count, sampling interval.
280 FOR J=0 TO N STEP L :REM Write compressed data set.
290 PRINT#1,A(J)
300 NEXT J
310 CLOSE
320 L$="th"
330 IF L=2 THEN L$="nd" :REM Tell what you did.
340 IF L=3 THEN L$="rd"
350 PRINT "Every ";L;L$;" data value has been kept in the
    compressed data file (CDATA)."
360 PRINT "The original data set can be reconstructed to
    an accuracy of +/-";E
```

LISTING3
Contributed by Robert J. Sciamanda
"Another Approach to Data Compression," by Robert J. Sciamanda, page 137.

```
10 OPEN "I",#1,"CDATA" :REM Compressed data.
20 INPUT#1,N,L  :REM Get count, sampling inverval.
30 K=INT(N/L):DIM B(K)
40 FOR I=0 TO K :REM Get compressed data.
50 INPUT#1,B(I)
60 NEXT I
70 CLOSE
80 OPEN "O",#1,"RDATA" :REM Create reconstructed data.
90 PRINT#1,N              :REM Write data count.
100 W=3.141593/L
110 FOR I=0 TO N          :REM Reconstruction
120 IF I MOD L = 0 GOTO 190 :REM Branch at sampled values.
130 G=0
140 FOR J=0 TO K          :REM The Nyquist sum.
150 M=W*(I-J*L)
160 G=G+B(J)*SIN(M)/M
170 NEXT J
180 GOTO 200     :REM Sum done; store this value.
190 G=B(I/L)
```

continued

```
200 PRINT#1,G    :REM Write reconstructed value to file.
210 NEXT I       :REM Go reconstruct next value.
220 CLOSE        :REM Done
230 PRINT "The reconstructed data file is RDATA"
```

LISTING4
Contributed by Robert J. Sciamanda
"Another Approach to Data Compression," by Robert J. Sciamanda, page 137.

```
10 OPEN "I",#1,"DATA"    :REM Original data file.
20 OPEN "I",#2,"RDATA"   :REM Reconstructed data file.
30 PRINT "  DATA          RDATA       Error"
40 IF EOF(1) THEN CLOSE: END
50 INPUT#1,A     :REM Get original data value.
60 INPUT#2,B     :REM Get reconstructed value.
70 ER=ABS(B-A)   :REM Calculate error.
80 PRINT USING "#.#####^^^^  #.#####^^^^  #.####";A,B,ER
90 GOTO 40
```

LISTING5
Contributed by Robert J. Sciamanda
"Another Approach to Data Compression," by Robert J. Sciamanda, page 137.

```
10 SCREEN 0,0,0 :REM Text screen
20 SZ=4-INT(-(640+7)*200/32) :REM Size of graphics array.
30 DIM SC(SZ)             :REM To hold graphics screen.
40 VRES=200: HRES=640
50 MIDY=INT((VRES-1)/2) :REM Vertical offset for X-axis
60 YES=(1=1)
70 NO=(1=0)
80 SS=NO                  : REM Screen not saved yet
90 FILES
100 LINE INPUT "Name the input file ";FI$
110 IF FI$=NU$ THEN END
120 OPEN FI$ FOR INPUT AS 1
130 INPUT #1,N
140 PRINT FI$; " contains ":N+1; "values"
150 INPUT #1,Y
160 MINY=Y: MAXY=Y
170 FOR K=1 TO N
180 INPUT #1,Y
190 IF Y>MAXY THEN MAXY=Y
200 IF Y<MINY THEN MINY=Y
210 NEXT K
220 CLOSE
230 PRINT "Values range from ";MINY; "to "; MAXY
240 PRINT "Press any key to continue";
250 WHILE INKEY$=NU$: WEND
260 YSCALE=(VRES-1)/ABS(MAXY-MINY)
270 XSCALE=(HRES-1)/N
280 CLS
290 SCREEN 2    :REM Graphics screen
300 IF SS THEN PUT (0,0),SC :REM Restore screen if it has
    been saved previously.
310 LINE (0,MIDY)-(HRES-1,MIDY),1        :REM Draw X-axis
320 OPEN FI$ FOR INPUT AS 1
330 INPUT #1,N
340 INPUT#1,Y
350 PSET (0,(Y-MINY)*YSCALE)    :REM Plot first point.
360 FOR X=1 TO N
370 INPUT #1,Y
380 LINE -(X*XSCALE,(Y-MINY)*YSCALE) :REM Connect points
390 NEXT X
400 CLOSE
410 GET (0,0)-(639,199),SC
420 WHILE INKEY$=NU$: WEND  :REM Hold til key pressed.
430 SS=YES       :REM Screen has been saved.
440 SCREEN 0,0,0         :REM Go back to text screen.
```

```
450 GOTO 90
5000 DEF SEG=0
5010 PRINT "Color or mono display (c/m)? ";
5020 CM$=INPUT$(1)
5030 PRINT CM$
5040 WHICH=INSTR(1,"CcMm",CM$)
5050 ON WHICH+1 GOTO 5010,5070,5070,5130,5130
5060 END
5070 POKE &H410,(PEEK(&H410) AND &HCF) OR &H10
5080 SCREEN 1,0,0,0
5090 SCREEN 0
5100 WIDTH 40: WIDTH 80
5110 LOCATE ,,1,6,7
5120 STOP
5130 POKE &H410,(PEEK(&H410) OR &H30)
5140 SCREEN 0
5150 WIDTH 40
5160 WIDTH 80
5170 LOCATE ,,1,12,13
```

```
IFP       TXT...
LISTING1     ...
LISTING2     ...
LISTING3     ...
LISTING4     ...
LISTING5     ...
POLY      BAS...
```

March

LIST1.TXT
Contributed by: Brian Edgington
"Intalling Memory-Resident Programs with C", by Brian Edgington, March 1987, page 129.

```
#include <dos.h>
main()
{
extern int _TSIZE;      /* size of program in paragraphs */
   union REGSS input,output;

       input.x.ax = 0x3112;     /* 31 -> AH and Return Code */
       input.x.dx = _TSIZE;     /* program size (Lattice)   */
       intdoss(&input,&output); /* function call 31  */
        }
```

LIST2.TXT
Contributed by: Brian Edgington
"Intalling Memory-Resident Programs with C", by Brian Edgington, March 1987, page 129.

This project started as a challenge to make a friend's calculator program load and remain resident in memory on an IBM PC. Making a program written in assembly langauge stay resident has been presented in many articles and books, but writing the tools to make a C program resident was a new adventure. I developed all the examples in this article with Lattice's C Compiler version 3.0 and Microsoft's Macro Assembler 4.0. I have tried to make everything as portable as possible, but I'm sure that some modification will have to be made for different compilers and languages. In the listings, I have noted any compiler-dependent variables. (Editor's note: William Claff's article, "xxxxxx" on page ?? contains additional information on the topic of DOS extension via memory-resident programs.)

WHAT IS A RESIDENT PROGRAM?

DOS uses a set of pointers called Storage Blocks to keep track of allocated and unallocated memory in the system. For each loaded program, these pointers indicate the address its PSP (Program Segment Prefix) the program's length in segments. There is also a flag that indicates whether or not the memory pointed to by the Storage Block is allocated. When a program module is loaded and executes an INT 27H (terminate but stay resident) or DOS function 31H (keep process), COMMAND.COM makes sure that this program becomes a part of DOS. This means that the Storage Block, PSP, and the program module remain in memory and are not reallocated.

The principles behind making a program resident seems to be straightforward, just find the length of the program, shove it into a register and call a documented function. DOS Function 31H requests the program size in paragraphs be placed in DX and the return code if any in AL.

As demonstrated by the program shown in listing 1, it is a simple matter to make a program resident. If you have a utility like Norton's SI or SMAP you can verify that the program is indeed resident by looking at the location of the next program to be loaded address. You can also examine the amount of free memory displayed by the CHKDSK utility before and after running the program.

Usually, we want to write a program that is more helpful than just taking up memory. Specifically, we want to write a program

continued

that responds to a system interrupt, and in doing so, supplies us with some sort of information. It should also be "well behaved" and operate within the constraints of DOS.

The design of this installation system had several primary goals:

* Modular design for universal application.
* Optimum memory usage.
* Correct processing of interrupts.

Modular design means that I can, with minor revision, make this program load any module that meets with the requirements for a resident, interrupt processing program. To determine these requirements, I made a careful analysis of what my compiler did to a program, and what my linker did to the object modules supplied to it. If you are using a compiler and/or linker other than the ones I used, these requirements may be different. Listing 2 is an example of a completed sample system. Since we have little control right now over anything that happens above main(), we'll start there and analyze what happens. Refer to listing 3, which is a dissasembled version of the top-level code in listing 2.

Cpush() and cpop() are two routines we'll create later to help us get into and return from the interrupts. Since main() is really just another function called by the compiler's entry module which is what is really loaded by EXEC, the BP register is saved and then set to the new SP. This is a requirement of any functions called from another routine that might pass any information on the stack; it allows the functions to reference that information on the stack via the BP register while still permitting new data to be pushed on the stack as required.

Entry into a resident program should be designed so any parameters are passed in the DOS communications area or in registers, and not on the stack. Also, once a program module is installed in memory, we want to ignore the call to install(). Although this uses six bytes of memory, passing the address of the call to cpush() to the interrupt vector is the most efficient way to install the module. All function names are made common in a C compiler so we can create the new vector IP by:

```
nu_entry = (short)main + 6;
```

Casting main to a short keeps it consistent with the way the rest of the register structures are typed. nu_entry now points to the desired entry point in the program. Since we did not need to use the compiler-generated PUSH BP, and we are returning from an interrupt we can ignore the POP BP and the RET that the compiler put at the end of main.

The install() function is straightforward. In this example I borrowed an unused function call's vector to leave a signature or message to the calling program that we are already installed. To increase the safety of this routine, you could verify that the interrupt vector is filled with zeros first. If it is not, check another vector until one is found with no vector already installed. Alternately, you could indicate that the module is already installed by setting a flag in memory, but you would have to choose a byte that you are certain would not be used by some other routine.

Another method for routines that handle passed values (i.e. video calls, put and get char and string calls) would be to detect a certain value, and return an 'already installed' message to the installation program. Listing 4 shows a segment of code that you could modify to perform this method of signature detection.

The next task is to decide how to best utilize the memory taken up by the program. Since I used function call 31h instead of int 21h to terminate the program, loaded programs can exceed the 64k limit imposed by the latter. I can use .EXE programs with stack and data segments defined -- not just .COM programs. A .COM program uses as much memory as the machine has left when it is loaded; if the program is going to stay resident, it has to return its unused memory to the system.

I release the memory that contains the program's copy of the environment using routine d_env(). On entry, the ES and DS (and SS and CS for a .COM program) segment registers point to the PSP at offset 0. Listing 5 shows the code for d_env(). I load ES with the address of the segment containing the copy of the environment and call DOS function 49H (free allocated memory).

If the program is a .COM file, you can reduce its size using routine shrink() (see listing 6). This function sets the memory used by a program to the size of the program module in paragraphs. If you write .COM programs, be sure that you allocate stack area before calling this function. If you use shrink(), you should call it before calling d_env() so that the ES register contains the correct information for the call to function 4AH (modify allocated memory blocks). (You could modify the code to perform both operations with one call to increase the speed and reduce the size of the program.)

The last area I will cover concerning memory management is one that is heavily influenced with my familiarity with the Lattice compiler. This compiler uses a file to set up the segment registers, handle stack and memory allocations, report errors such as stack overflows, `handle command line arguments to change the stack size, redirect I/O, and some other incidental operations. The code for all this is found in the c.asm file -- its object module is in c.obj. This code is loaded before main() and cannot be efficiently deallocated by any means other than actually editing out unused portions of c.asm and recompiling the file. A knowledgeable programmer should be able to remove large portions of c.asm for many applications; I have reduced considerable space in mine.

INTERRUPTS

Now that I've shown how to load programs into memory and keep them resident, let's examine the available methods of processing the interrupts (keyboard, clock, etc.) and determine the best possible way to maintain 'nice' programs. My main concern is with the saving of registers and flags because of the amount of calls and subroutines normally found in a program written in C. (You can see an example of this in listing 1.)

Since we passed the address of cpush() to the interrupt vector, the first thing the program does when it is entered is a call to cpush(). This call pushes a return address on the stack, one that would not be there if the code was being generated in assembly language. This problem is repeated throughout the program so it must be handled very early on.

The three modules in listing 7 show one of the fastest and most efficient solutions I found. Upon entry into the program I call cpush(). This routine stores the short call return address, the interrupting program's return CS and IP, and the FLAGS that are pushed on the stack. It then stores the registers and segment registers in its own allocated memory. The short return address is then pushed back onto the stack and the function returns to the body of the program.

After the interrupt routine does its work (in the example given in listing 2, it prints "Hello world"), it calls cpop() to return to the interrupted program. The cpop() routine emulates a pop of all the registers that should have been pushed onto the stack upon entry into the interrupt handler, and then does an IRET. (For debugging purposes, I have also included the code for cpopt(), which is similar to cpop() except that cpopt() exits via a RET instruction.)

SUMMARY

This article demonstrates a very simple interrupt processing program that remains resident in memory. I plan to do more work in writing programs that process the keyboard and video interrupts. Any programs written that use these techniques should be written with proper attention to good program structure, and correct manipulation of pointers and addresses. This project

continued

turned out to be a lot more ambitious than I originally thought. The entry and exit routines posed the most problem, testing and debugging sometimes left the machine in a very corrupted state. Be certain that your C programs can pass lint before using them; remember, you are creating a extension of DOS. I did notice that including structures in a program compiled with Lattice increased the address of the entry point by three. It makes a call after main() to set up the memory for the structs and/or unions. I will be interested in feedback about improving any of these algorithms and techniques.

LIST3.TXT
Contributed by: Brian Edgington
"Intalling Memory-Resident Programs with C", by Brian Edgington, March 1987, page 129.

```
main()
{
0000              PUSH  BP
0001              MOV   BP,SP
0003  install();  CALL  install
0006  cpush();    CALL  cpush
      printf("Hello world\n");
0012  cpop();     CALL  cpop
0015              POP   BP
0016              RET
}
```

LIST4.TXT
Contributed by: Brian Edgington
"Intalling Memory-Resident Programs with C", by Brian Edgington, March 1987, page 129.

```
calling(program)
{
        union REGSS in,out;

        ......
        in.x.ax = 0x0088;
        intdoss(int_num,&in,&out);
        ........
}
```

```
int_num

        switch((int)in.x.ax){
        case 0x88:
                return installed
        case 0xXX
                do something else
```

LIST5.TXT
Contributed by: Brian Edgington
"Intalling Memory-Resident Programs with C", by Brian Edgington, March 1987, page 129.

```
; d_env is used to deallocate the memory used
; by a program's copy of the environment.

        TITLE       ENVIRONMENT DEALLOCATION
        SUBTTL      Copyright 1986 Brian Edgington
        NAME        D_ENV
        INCLUDE     DOS.MAC

        PSEG
        PUBLIC  D_ENV
        IF      LPROG
D_ENV   PROC    FAR
        ELSE
```

```
D_ENV   PROC    NEAR
        ENDIF

        PUSH    BP
        MOV     BP,SP
        MOV     ES,[DI+2CH]
        MOV     AX,4900H
        INT     21H
        POP     BP
        RET
D_ENV   ENDP
        ENDPS
        END
```

LIST6.TXT
Contributed by: Brian Edginton
"Intalling Memory-Resident Programs with C", by Brian Edginton, March 1987, page 129.

```
        TITLE     PROGRAM SHRINKER
        SUBTTL    Copyright 1986 Brian Edginton
        NAME      SHRINK
        INCLUDE   DOS.MAC

        PSEG
        PUBLIC  SHRINK
        IF      LPROG
SHRINK  PROC    FAR
        ELSE
SHRINK  PROC    NEAR
        ENDIF
```

```
        PUSH    BP
        MOV     BP,SP
        MOV     BX, ????    ; PROGRAM SIZE + ANY
                            ; STACK YOU NEED.
        MOV     AX,0X4A00   ; CALL MODIFY
                            ; ALLOCATED MEMORY
        INT     21H         ; BLOCKS ROUTINE.
        POP     BP
        RET
SHRINK  ENDP
        ENDPS
        END
```

LIST7.TXT
Contributed by: Brian Edginton
"Intalling Memory-Resident Programs with C", by Brian Edginton, March 1987, page 129.

```
; These routines emulate a push-all of the
; registers and segregs before an interrupt
; is executed, and then pop them in the
; same order.
; Note: cpopt is used for testing so we use
; a ret instead of a iret.


        TITLE   REGISTER MANIPULATION ROUTINES
        SUBTTL  Copyright 1986 by Brian Edginton
        NAME    STORE
        INCLUDE DOS.MAC

        DSEG
; Register and stack storage structure follows.
S       STRUC
        RAX     DW      ?
        RBX     DW      ?
        RCX     DW      ?
        RDX     DW      ?
        RSI     DW      ?
        RDI     DW      ?
        RDS     DW      ?
        RCS     DW      ?
        RSS     DW      ?
        RES     DW      ?
        RIP     DW      ?   ; Storage for 2 words
        RBP     DW      ?   ; pushed for iret
        RFL     DW      ?   ; and flags.
        RRT     DW      ?   ; Return address
                            ; pushed for call.
S       ENDS
STORAGE S       <0,0,0,0,0,0,0,0,0,0,0,0,0>
        ENDDS


        PSEG

        ;  name - cpush()
        ;  Pushes all the registers and segregs
        ;  and flags onto the stack.  Use with
        ;  cpop() to restore in correct order.

        PUBLIC  CPUSH
        IF      LPROG
CPUSH   PROC    FAR
        ELSE
```

```
CPUSH   PROC    NEAR
        ENDIF

        POP     STORAGE.RRT
        POP     STORAGE.RIP         ; SAVE RETI IP,
        POP     STORAGE.RCS         ; SAVE RETI CS,
        POP     STORAGE.RFL         ; SAVE THE FLAGS,
        MOV     STORAGE.RAX,AX      ; AND TUCK AWAY
        MOV     STORAGE.RBX,BX      ; ALL REGISTERS.
        MOV     STORAGE.RCX,CX
        MOV     STORAGE.RDX,DX
        MOV     STORAGE.RDI,DI
        MOV     STORAGE.RSI,SI
        MOV     STORAGE.RDS,DS      ; GET THE SEGMENT
        MOV     STORAGE.RSS,SS      ; REGISTERS.
        MOV     STORAGE.RES,ES
        PUSH    STORAGE.RRT         ; PUT CALL RETURN
                                    ; ADDRESS ON STACK.
        RET
CPUSH   ENDP

        PUBLIC  CPOP
        IF      LPROG
CPOP    PROC    FAR
        ELSE
CPOP    PROC    NEAR
        ENDIF

        PUSH    STORAGE.RFL         ; RESTORE FLAGS.
        PUSH    STORAGE.RCS         ; RESTORE CS,
        PUSH    STORAGE.RIP         ; THEN IP.
        MOV     ES,STORAGE.RES
        MOV     DS,STORAGE.RDS
        MOV     SI,STORAGE.RSI
        MOV     DI,STORAGE.RDI
        MOV     DX,STORAGE.RDX
        MOV     CX,STORAGE.RCX
        MOV     BX,STORAGE.RBX      ; READY FOR IRET
        MOV     AX,STORAGE.RAX      ; WITH OLD IP AND
        IRET                        ; STUFF ON STACK.
CPOP    ENDP

        PUBLIC  CPOPT
        IF      LPROG
CPOPT   PROC    FAR
        ELSE
```

continued

```
CPOPT   PROC      NEAR
        ENDIF
        PUSH      STORAGE.RFL      ; RESTORE FLAGS.
        PUSH      STORAGE.RCS      ; RESTORE CS,
        PUSH      STORAGE.RIP      ; THEN IP.
        MOV       ES,STORAGE.RES
        MOV       DS,STORAGE.RDS
        MOV       SI,STORAGE.RSI
        MOV       DI,STORAGE.RDI
        MOV       DX,STORAGE.RDX
        MOV       CX,STORAGE.RCX
        MOV       BX,STORAGE.RBX   ; READY FOR IRET
        MOV       AX,STORAGE.RAX   ; WITH OLD IP AND
        RET                        ; STUFF ON STACK.

CPOPT   ENDP
        ENDPS
        END
```

BIXMODEM.INC
Contributed by Barry Nance
"Best of BIX," PC.BIX/SOURCE.CODE #28 from barryn (Barry Nance), March 1987, page 311.

```
again
=========================
pc.bix/source.code #28, from barryn, 9661 chars, Thu Jun 26 19:48:15 1986
-------------------------
TITLE: BIXMODEM.INC

{                                                                    }
{                                                                    }
{                                                                    }
{ BIXMODEM.INC  Ymodem procedures for use with BIX.PAS               }
{                                                                    }
{                                                                    }
{      Program and all Supporting Materials Copyright                }
{      (c) 1985 Barry R. Nance                                       }
{               17 Pease Street                                      }
{               Wilbraham, Massachusetts 01095                       }
{               (413) 596-4031                                       }
{                                                                    }
{                                                                    }


Var  CRCWork : Integer;
     CRC     : Integer;

Function PartialCrc (OldCRC:Integer; C:Char) : Integer;
         {done in 80x8x assembler for speed}
Begin
  CRCWork := OldCRC;

  INLINE( $8A / $46 / $04 /          (* Mov    Al,[Bp+4]    *)
          $8B / $1E / CRCWork /      (* Mov    Bx,CRCWork   *)
          $B9 / $08 / $00 /          (* Mov    Cx,8         *)
{Oloop:}  $D0 / $E0 /                (* Shl    Al,1         *)
          $D1 / $D3 /                (* Rcl    Bx,1         *)
          $73 / $04 /                (* Jnc    Iloop        *)
          $81 / $F3 / $21 / $10 /    (* Xor    Bx,$1021     *)
{Iloop:}  $E2 / $F4 /                (* Loop   Oloop        *)
          $89 / $1E / CRCWork )      (* Mov    CRCWork,BX   *);

  PartialCRC := CRCWork;
  End;


Procedure ReceiveXMODEM (XName : Str20);
Const
    SOH   = #$01;
    STX   = #$02;
    EOT   = #$04;
    ACK   = #$06;
    NAK   = #$15;
    C_Ch  = 'C';
```

```
Type
    YrecDef     = Array [1..1024] of Char;
    XrecDef     = Array [1..128]  of Char;

Var
    Xrec        : XrecDef;
    Yrec        : YrecDef;
    XFile       : File of XrecDef;

    XSub        : Integer;
    ErrCnt      : Integer;
    BlockError  : Boolean;
    CurrBlock   : Integer;
    EOTdetected : Boolean;
    BlockLength : Integer;
    Duplicate   : Boolean;
    GetOutFlag  : Boolean;
    FirstNAK    : Boolean;


      Function Abort : Boolean;
      Begin
        Abort := False;

        If ErrCnt > 10 then
           Begin
             HighVideo;
             Write (^G);
             Write (
'Ten errors have occurred on this block.  Continue (Y/N)? ');
        LowVideo;
        Repeat Read(kbd, Key) Until UpCase(Key) in ['N', 'Y'];
        Writeln (Key);
        If UpCase(Key) = 'N' then
           Begin
             Abort      := True;
             GetOutFlag := True;
             End
        Else
           ErrCnt := 0;
        End;

    End;



  Procedure SendNAK;
  Begin
    PurgeBuffer;

    If Duplicate then Exit;

    SendChar(NAK);
    Writeln ('Requesting re-transmission of block # ', CurrBlock);
    ErrCnt      := Succ(ErrCnt);
    BlockError  := True;
    End;



  Procedure SendACK;
  Begin
    SendChar(ACK);
    ErrCnt := 0;
    End;



  Procedure ReceiveSOH;
  Begin
    ReceiveChar (10, Ch, TimedOut);
```

continued

```
 If Ch = EOT then
    Begin
      EOTdetected := True;
      SendACK;
      Exit;
      End;

 If Ch = C_Ch then
    If CurrBlock = 1 then
       ReceiveChar (10, Ch, TimedOut);

 If TimedOut then
    If CurrBlock = 1 then
       If FirstNAK then
          Begin
            FirstNAK := False;
            SendChar (NAK);
            ReceiveChar (10, Ch, TimedOut);
            End;

 If (TimedOut)
       or
    ((Ch <> SOH) And (Ch <> STX))  then
    Begin
      If TimedOut then
         Writeln ('Timed out on SOH/STX.')
      Else
         Writeln ('1st char not SOH/STX.');
      SendNAK;
      End
Else
    If Ch = STX then
       BlockLength := 1024
    Else
       BlockLength := 128;
End;

Procedure ReceiveBlockNum;
Var    Blk     : Byte;
       PrevBlk : Byte;
       FirstCh : Char;
Begin
  If BlockError then Exit;

  Duplicate := False;
  Blk       := CurrBlock Mod 256;
  PrevBlk   := (CurrBlock - 1) Mod 256;
  ReceiveChar (1, Ch, TimedOut);
  FirstCh := Ch;

  If (TimedOut) or (Ord(Ch) <> Blk)  then
     If Ord(Ch) <> PrevBlk then
        Begin
          SendNAK;
          If TimedOut then
             Writeln ('Timed out on block number.')
          Else
             Writeln ('Block number error (calcd = ', Blk, ').');
          Exit;
          End;

  ReceiveChar (1, Ch, TimedOut);
  Blk     := 255 - Blk;
  PrevBlk := 255 - PrevBlk;

  If (TimedOut) or (Ord(Ch) <> Blk) then
     If Ord(Ch) <> PrevBlk then
        Begin
          SendNAK;
          If TimedOut then
             Writeln ('Timed out on complement.')
          Else
             Writeln ('Complement error (calcd = ', Blk, ').');
          Exit;
          End;
  If Ord(Ch) = PrevBlk then
     If Ord(FirstCh) = CurrBlock Mod 256 then
        Duplicate := True;

  End;

Procedure ReceiveDataBlock;
Begin
  If BlockError then Exit;
  OverrunError := False;


  Repeat
    XSub := Succ(XSub);
    ReceiveChar (1, Ch, TimedOut);

    If Not TimedOut then
       Begin
         Yrec [XSub] := Ch;
         If BlockLength = 1024 then
            CRC := PartialCRC (CRC, Ch);
         End;

    Until (TimedOut) or (XSub = BlockLength) or (OverrunError);


  If (TimedOut) or (OverrunError) then
     Begin
       SendNAK;
       If TimedOut then
          Writeln ('Timed out waiting for data.')
       Else
          Writeln ('Overrun error occurred.');
       OverrunError := False;
       End;
  End;


 Procedure ReceiveCheckSum;
 Var    ChkSum : Byte;
 Begin
   If BlockError then Exit;
   ReceiveChar (1, Ch, TimedOut);
   ChkSum := 0;
   For XSub := 1 to 128 Do
       ChkSum := ChkSum + Ord(Yrec[XSub]);
   If (TimedOut) or (ChkSum <> Ord(Ch)) then
      Begin
        SendNak;
        If TimedOut then
           Writeln ('Timed out on checksum.')
        Else
           Writeln (
           'Checksum error (is ', Ord(Ch), '; should be ', ChkSum, ').');
        End;
   End;


 Procedure ReceiveCRC;
 Var
   CRCin  : Integer;

 Begin
   If BlockError then Exit;

   ReceiveChar (1, Ch, TimedOut);

   If Not TimedOut then
      Begin
        CRC   := PartialCRC (CRC, Ch);
        CRCin := ord(Ch) * 256;
        ReceiveChar (1, Ch, TimedOut);
        If Not TimedOut then
```

continued

```
                Begin
                  CRC   := PartialCRC (CRC, Ch);
                  CRCin := CRCin + ord(Ch);
                  End;
             End;

        If (TimedOut) or (CRC <> 0) then
           Begin
             SendNAK;
             If TimedOut then
                Writeln ('Timed out on CRC.')
             Else
                Writeln (
                'CRC error (is ', CRCin, '; should be ', CRC, ').');
             End;
        End;


Procedure GetXMODEMBlock;
Begin
  If Keypressed then
     Begin
       GetKey (Key, Extended);
       If Key = Chr(27) then
          Begin
            GetOutFlag := True;
            Exit;
            End;
       End;

  BlockError := False;
  ReceiveSOH;

  If EOTdetected then Exit;

  ReceiveBlockNum;

  XSub := 0; CRC := 0;
  ReceiveDataBlock;

  If BlockLength = 1024 then
     ReceiveCRC
  Else
     ReceiveCheckSum;

  If Not BlockError then
     Begin
       SendACK;
       If Not Duplicate then
          Begin
            Writeln ('Block # ', CurrBlock, ' received.');
            If BlockLength = 128 then
               Begin
                 Move  (Yrec[1], Xrec[1], 128);
                 Write (XFile, Xrec);
                 End
            Else
               Begin
                 For XSub := 1 to 8 Do
                     Begin
                       Move  (Yrec[((XSub - 1) * 128) + 1], Xrec[1], 128);
                       Write (XFile, Xrec);
                       End;
                 End;
            CurrBlock := Succ(CurrBlock);
            End;
       End;
  End;


Begin                            {of ReceiveXMODEM}
 If XName = '' then Exit;

 Assign  (XFile, XName);
 Rewrite (XFile);
 Writeln ('File ', XName, ' is being received.');
 Writeln;

 UpdateUART (8, 'N', 1);
 PurgeBuffer;
 SendChar(C_Ch);

 FirstNAK       := True;
 OverrunError   := False;
 DoingXMODEM    := True;
 XSub           := 0;
 ErrCnt         := 0;
 CurrBlock      := 1;
 BlockError     := False;
 EOTdetected    := False;
 Duplicate      := False;
 GetOutFlag     := False;

 Repeat
   GetXMODEMBlock;
   Until (Abort) or (EOTdetected) or (GetOutFlag);

 If GetOutFlag then
    Begin
      Close   (XFile);
      Erase   (XFile);
      Writeln ('ERROR--reception of ', XName, ' cancelled.  File erased.');
      End
 Else
    Begin
      Close   (XFile);
      Writeln;
      Writeln (XName, ' successfully received.');
      End;

 DoingXMODEM:= False;
 UpdateUART (7, 'E', 1);

 End;

Read:
```

IMAGEIO.C
Contributed by: Chuck McManis
"Low-Cost Image Processing," by Chuck McManis, March 1987, page 191

```
/*
 *              imageio.c
 *
 * These routines provide the base level image I/O routines. There is a
 * readimage() and writeimage() routine. Both require only a pointer to
 * a memory array and a pointer to a file name.
 *
 */

#include <exec/types.h>         /* The UBYTE and USHORT types are here  */
#include <stdio.h>              /* The standard C I/O functions         */
#include <fcntl.h>              /* The Level 1 file I/O constants       */

/*
 * Function : SetPixel
 * This function will set a pixel in the image to the given value.
 * It is passed four values; the first is a pointer to the
 * image array, the second through fourth are the row, column and value
 * of the pixel, which are all integers.
 */

void
SetPixel(image,col,row,val)
```

continued

```
UBYTE   image[];            /* An array of pixels                            */
int     col,                /* The pixel x coordinate or column             */
        row,                /* The pixel y coordinate or row                */
        val;                /* a value between 0 and 15 for the pixel        */

{
  int   temp,               /* a temporary value                              */
        index,              /* The index into the array                       */
        shift;              /* shift factor (4 if pixel even, 0 if it is odd) */

  /* Since two pixels are contained in a byte the 640 pixel line is really
   * 320 bytes wide, and the column value divided by two is the byte containing
   * the pixel we want. If column is even the pixel is in the left half of the
   * byte and if column is odd the pixel is in the right half.
   */

  shift = (col % 2) ? 0 : 4 ;
  index = (row * 320) + (col / 2);
  temp = val << shift;
  if ((index < 0) || (index > 127999)) /* Index checking */
    printf("Error! Bad row and column passed to SetPixel.\n");
  else
    image[index] = (image[index] & (0x0f0 >> shift)) + temp;
}

/*
 * Function : Pixel
 * This function will return the value of the pixel in the image. It is
 * passed four values; the first is a pointer to the image array, the
 * second through fourth are the row, column and value of the pixel, which
 * are all integers.
 *
 */

int
Pixel(image,col,row)

UBYTE   image[];            /* An array of pixels                            */
int     col,                /* The pixel x coordinate or column             */
        row;                /* The pixel y coordinate or row                */

{
  int   temp,               /* a temporary value                              */
        index,              /* The index into the array                       */
        shift;              /* shift factor (4 if pixel even, 0 if it is odd) */

  /* This is the same calculation as in SetPixel above */

  temp = 0;
  shift = (col % 2) ? 0 : 4 ;
  index = (row * 320) + (col / 2);
  if ((index < 0) || (index > 127999)) /* Index checking */
    printf("Error! Illegal values passed to Pixel(%d,%d).\n",col,row);
  else
    temp = (image[index] >> shift) & 0x0f;
  return(temp);
}

/*
 * Function : ReadImage
 * This function will read in the 4 bitplanes from the file specified and
 * store them as 4 bit pixels in the image array. It returns zero if it
 * was successful, and a negative number if it detected an error.
 * Errors include :
 *      -1      Couldn't open 'filename'
 *      -2      Didn't read enough pixel data.
 *      -3      Didn't read enough colormap data.
 */

int
ReadImage(filename, image, colormap)

char    *filename;          /* A pointer to a filename string       */
UBYTE   *image;             /* A pointer to a 128000 byte array     */
USHORT  *colormap;          /* An array of color map entries        */
```

```
{
  short i,j,k;
  int   n,fh,             /* Byte count, File Handle                        */
        error,            /* indicator that an error occurred during read   */
        ishft,pshft;      /* Some shift factors for manipulating bits.      */
  UBYTE pixels[80];       /* 640 bits worth (one line) of pixel data.       */

  /* Open the input file */

  fh = open(filename,O_RDONLY);
  if (fh == -1) return(-1); /* Return error if it couldn't be opened */
  error = 0;

  printf("Reading in source image from file %s ... ",filename);
  for (i=0; i<4; i++) { /* Four bit planes */
    for (k=0; k<400; k++) { /* 400 lines */
      n = read(fh,pixels,80); /* Read in a row of pixels */
      if (n == 80) {
        for (j=0; j<640; j++) {         /* Unpack the pixels                */
          ishft = (j % 2) ? i+4 : i;    /* the image's byte shift factor */
          pshft = (7-j%8);              /* The pixel's byte shift factor */
          *(image+(j/2)+(k*320)) &= ~(1 << ishft); /* Clear old bit      */
          *(image+(j/2)+(k*320)) |= ((pixels[j>>3] >> pshft) & 1) << ishft;
        }
      }
      else error = -1; /* If we didn't get enough data it is an error */
    }
  }
  if (error == 0) {
    j = read(fh,(UBYTE *)colormap,32); /* Read in the color map */
    if (j != 32) error = -1;
  }
  printf("Done.\n");
  close(fh);
  return(error);
}

/*
 * Function : WriteImage
 * This function will write out the image array to the specified file. It
 * converts the 4 bit/pixel format of the image array to the bit plane
 * format used by the iff conversion programs. It returns zero if it was
 * successful and a negative number if it detected an error.
 * Errors include :
 *      -1      Couldn't open file
 *      -2      Unexpected EOF (probably the disk was full)
 *      -3      Unexpected EOF while writing colormap
 */

int
WriteImage(filename, image, colormap)

char    *filename;        /* A pointer to a filename string        */
UBYTE   *image;           /* A pointer to a 128000 byte array      */
USHORT  colormap[];       /* An array of color map entries         */

{
  short i,j,k;
  int   n,fh,             /* byte count, File Handle                        */
        error,            /* indicator that an error occurred during read   */
        ishft,pshft;      /* Some shift factors for manipulating bits.      */
  UBYTE pixels[80];       /* 640 bits worth (one line) of pixel data.       */

  /* First we open the file */
  fh = open(filename,O_WRONLY+O_CREAT); /* Write only, and create it */
  if (fh == -1) return(-1);
  error = 0;

  /* Now write out the image */

  printf("Writing the processed image to file %s ... ",filename);
  for (i=0; i<4; i++) {                 /* Four bit planes                  */
    for (k=0; k<400; k++) {             /* 400 lines                        */
      for (j=0; j<640; j++) {           /* Unpack the pixels                */
```

continued

```
        ishft = (j % 2) ? 4+i : i;   /* The bit number in the image byte   */
        pshft = 7 - (j % 8);         /* Pixel shift value for pixels array */
        pixels[j>>3] &= ~(1 << pshft);    /* clear previous bit            */
        pixels[j>>3] |= ((*(image+(j/2)+(k*320)) >> ishft) & 1) << pshft;
      }
      n = write(fh,pixels,80);  /* Write out this line of bits */
      if (n < 80) error = -2;
    }
  }
  if (error == 0) {
    j = write(fh,(UBYTE *)colormap,32); /* write out the color map */
    if (j < 32) error = -3;
  }
  printf("Done.\n");

  close(fh);            /* Clean up after ourselves */
  return(error);        /* And return */

}
```

EDGE.C
Contributed by: Chuck McManis
"Low-Cost Image Processing," by Chuck McManis, March 1987, page 191

```
/*
 *              edge.c
 *
 * This program will read in an image file, apply a simple edge detection
 * algorithm to it and then write out the resulting file. It demonstrates
 * the use of color on the output file to indicate edges.
 *
 * Usage edge input.image output.image threshold
 *
 * At all pixels where there is a difference equal to or greater than
 * "threshold" that pixel will be set to red. Other pixels left intact.
 *
 * This program is based on the Sobel edge detection algorithm, it uses the
 * fact that objects in an image are usually delineated by sharp changes in
 * intensity. The image is processed by picking 8 adjacent pixels and treating
 * them as a 3 X 3 array. The array can be represented as follows :
 *
 *      | a b c |
 *      | d e f |
 *      | g h i |
 *
 * And there are four unique straightline paths through this array which pass
 * through the center pixel. They can be represented as :
 *
 *      g -> e -> c
 *      d -> e -> f
 *      a -> e -> i
 *      b -> e -> h
 *
 * The algorithm treats the three pixels as points on a line described by the
 * function : Intensity = M * x + C. The parameter of interest is the slope
 * of the function M. The sharper the transition in intensity, the larger the
 * value of the slope. This is compared to the threshold and if it exceeds it
 * the pixel e is considered to lie on an edge and is marked as such.
 *
 */

#include <exec/types.h>
#include <exec/memory.h>
#include <stdio.h>
#include <fcntl.h>

/* This array describes the relative offsets of adjacent pixels that make up
 * the edge of interest.
 */
static int edges[] =  {
```

```
                         -1, 1,  1,-1,  /* g and c */
                         -1, 0,  1, 0,  /* d and f */
                         -1,-1,  1, 1,  /* a and i */
                          0,-1,  0, 1   /* b and h */
                        };


void main(argc,argv)
int argc;
char *argv[];

{
  UBYTE *image,*simage; /* An array for our image                       */
  USHORT colors[16];    /* The color map                                */
  int   i,ii,j,         /* Some counters                                */
        thresh,         /* The edge threshold                           */
        edgepixels,     /* The number of edge pixels found              */
        x,y,m,p1,p2,    /* Image coordinates                           */
        x1,x2,y1,y2;    /* Edge boundary coordinates                   */


  printf("Simple edge analysis program.\n");
  if (argc != 4) {
    printf("usage is Edge infile outfile threshold\n");
    exit(10);
  }
  thresh = atoi(argv[3]);
  if ((thresh < 0) || (thresh > 15)) {
    printf("Illegal threshold value, use a number between 0 and 15\n");
   exit(10);
  }
  printf("Using Threshold value %d.\n",thresh);

  /* Buffer for one image + 4 lines. The algorithm below will store the
   * resulting image into the image buffer - 4 lines. That way we do not
   * need to allocate two complete image buffers. The allocate call will
   * also set the array to zero.
   */
  image = (UBYTE *)AllocMem(129280,MEMF_CLEAR);

  if (image == NULL) {
    printf("Sorry couldn't allocate the image buffer! \n");
    exit(10);
  }
  simage = image + 1280; /* Source image resides four lines below image */

  /* First we read in the source image */
  i = ReadImage(argv[1],simage,colors);
  if (i != 0) {
     printf("Error reading in the source image.\n");
     FreeMem(image,129280);
     exit(10);
  }

  /* Now do what ever image processing we wish on the image data */

  printf("Processing the source image \n");
  edgepixels = 0;
  /* First copy the first row of pixels to the destination */
  for (i=0; i<640; i++) {
    j = Pixel(simage,i,0) - 1;
    if ((j < 0) || (j > 14)) j = 0;
    SetPixel(image,i,0,j);
  }
  for (y=1; y<399; y++) {
    printf("Line %d\x0d",y);
    /* Move a line from the Source image to the Destination Image */
    for (i=0; i<640; i++) {
      j = Pixel(simage,i,y) - 1;
      if ((j < 0) || (j > 14)) j = 0;
      SetPixel(image,i,y,j);
    }
    /* Now analyze each pixel in this line */
    for (x=1; x<639; x++) {
      /* (x,y) is the center pixel in a 3X3 square */
```

continued

```
      chkabort();       /* Added in case the user wants to abort */
      for (j=0; j<4; j++) { /* Check for four types of edges */
        /* Note the indirection through the color map since Pixel returns
         * the colormap entry this pixel uses, the colormap actually
         * contains its intensity. Also since all color map entries are
         * shades of gray R = G = B and the intensity is logically ANDed
         * to mask off the R and G components leaving only the B value
         * which will always be between 0 and 15.
         */
        x1 = x + edges[j*4];
        y1 = y + edges[j*4+1];
        x2 = x + edges[j*4+2];
        y2 = y + edges[j*4+3];
        p1 = colors[Pixel(simage,x2,y2)] & 0xf;
        p2 = colors[Pixel(simage,x1,y1)] & 0xf;
        m = abs(p1 - p2);
        if (m > thresh) {
          edgepixels++;
          SetPixel(image,x,y,15);
          break;
        } /* if we crossed the threshold */
      } /* for each value of j */
    } /* For x */
  } /* for y */
  printf("\n Done.\n");
  printf(" Set the value for %d edge pixels\n",edgepixels);
  /* Now we fix up the color map because we have shifted all of the pixels
   * down by one in the color map to make room for our edge color (red)
   */
  for (i=1; i<15; i++) colors[i-1] = colors[i];
  colors[15] = 0x0f00;  /* Set edge pixels to red */
  /* Then we write out the image */
  i = WriteImage(argv[2],image,colors);
  if (i != 0) printf("Error writing the output file!\n");
  FreeMem(image,129280);
  exit(i);
}
```

BASIC.LST
Contributed by: Brian Wichmann and David Hill
"Building a Random-Number Generator," by Brian Wichmann and David Hill, March 1987, page 127

```
10 PRINT "whbasic - 840930"
20 PRINT "basic version of Wichmann Hill generator"
30 REM J C Nash
40 REM x, y and z must be seeded as per article
50 PRINT "provide 3 integers as seeds to the generator"
60 PRINT "seed x=";
70 INPUT X
80 REM adjuust to be in range [0, 30269]
90 IF X = 0 THEN 120
100 LET X=X+30269
110 GOTO 90
120 IF X=30269 THEN 160
130 LET X=INT (X-30269)
140 GOTO 120
150 REM note use of int to ensure integer seed
160 PRINT "seed y=";
170 INPUT Y
180 REM adjust to be in range [0, 30307]
190 IF Y = 0 THEN 220
200 LET Y = Y+30307
210 GOTO 190
220 IF Y=30307 THEN 260
230 LET Y = INT (Y-30307)
240 GOTO 220
250 REM note use of int to ensure integer seed
260 PRINT "seed z=";
270 REM ADJUST TO BE IN RANGE  [O, 30323]
280 IF Z = 0 THEN 310
290 LET Z=Z+30323
300 IF Z=30323 THEN 350
320 LET Z=INT (Z-30323)
```

```
330 GOTO 310
340 REM  note use of int to ensure integer seed
350 INPUT Z
360 PRINT
370 PRINT "how many numbers are to be generated";
380 INPUT N
390 FOR I = 1 TO N
400 GOSUB 1000
410 PRINT
420 PRINT "current values -- x=";X;" y=";Y;" x=";Z
430 PRINT "   random fraction =";R
440 NEXT I
450 STOP
1000 REM comput next member of pseudo-random sequence
1010 LET X1=INT(X/177)
1020 LET X2=X-177*X1
1030 LET X=171*X2-2*X1
1040 IF X < 0 THEN LET X=X+30269
1050 LET Y1=INT (Y/176)
1060 LET Y2=Y-176*Y1
1070 LET Y=172*Y2-35*Y1
1080 IF Y < 0 THEN LET Y =Y+30307
1090 LET Z1=INT(Z/178)
1100 LET Z2=Z-178*Z1
1110 LET Z=170*Z2-63*Z1
1120 IF Z < 0 THEN LET Z=Z+30323
1130 REM combine generators to give function
1140 LET T=X/30269+Y/30307+Z/30323
1150 LET R=T-INT(T)
1160 REM get fractional part of t only
1170 RETURN
```

RANDOM.LST
Contributed by: Brian Wichmann and David Hill
"Building a Random-Number Generator," by Brian Wichmann and David Hill, March 1987, page 127

```
program...

var
   x, y, z: integer; { global seeds }
. . .
funtion random: real;
    var
        temp: real;
    begin
    { first generator }
    x := 171 * (x mod 177) - 2 * (x div 177);
    if x < 0 then
       x := x + 30269;
    { second generator }
    y :=172 * (y mod 176) - 35* (y div 176);
    if y < 0 then
       y :=y + 30307
    { third generator }
    z := 170 * (z mod 178) - 63* (z div 178);
    if z  < 0 then
       z := z + 30323
    { combine to give function value }
    temp := x/30269.0 + y/30307.0 + z/30323.0;
    random := temp - trunc(temp)
    end;
    ...
begin

{ initializse seeds.  For production runs, different
 values (between 1 and 30000) should be used each time,
preferably by some automatic method such as from date
and time readings if available }
x :=1; y := 10000; z:= 3000;
...
end
```

continued

LISPTEST.DOC
Contributed by William G. Wong
"PC Scheme: A Lexical LISP," by William G. Wong, March 1987, page 223

```
)
;; BYTE TI Scheme Benchmark Source     5-20-86 WGW

;; Time Test
(define (time-function function)
        (gc)                            ;; make sure system is consistent
        (let ((start-time (runtime)))
             (function)
             (/ (- (runtime) start-time) 100.0)
        )
)

(define (time-test function)
        (gc)                            ;; make sure system is consistent
        (let ((start-time (runtime)))
             (loop-test function 5000)
             (/ (- (runtime) start-time) 100.0)
        )
)

;; Loop test to get function time into timable range
(define (loop-test function limit)
        (do ((i 1 (1+ i)))
            ((>=? i limit))
            (function)
        )
)

;; Dummy function to test LOOP-TEST
(define (dummy))

;; List construction test
(define cons-var nil)
(define (cons-test) (cons cons-var cons-var))

;; Integer addition test
(define add-a 1)
(define add-b 2)
(define (add-test) (+ add-a add-b))

;; Integer multiplication test
(define mult-a 1)
(define mult-b 2)
(define (mult-test) (* mult-a mult-b))

;; Floating point addition test
(define fadd-a 1.2)
(define fadd-b 234324.3)
(define (fadd-test) (+ fadd-a fadd-b))

;; Floating point multiplication test
(define fmult-a 1.2)
(define fmult-b 234324.3)
(define (fmult-test) (* fmult-a fmult-b))

;; Assignment Test   (Load from variable and set global variable)
(define assign-a '(1 2 3))
(define (assign-test) (set! assign-a assign-a))

;; Local Assignment Test

(define (local-assign) (let ((x '())) (set! x '(1 2 3))))

;; List Indexing Test
(define (build-list length)
        (if (zero? length)
            '()
            (cons length (build-list (sub1 length)))
        )
)
```

```
(define list-a)
(set! list-a (build-list 128))

(define (list-index) (list-ref list-a 120))

;; Vector Index Test
(define vect-a)
(set! vect-a (make-vector 128 1))
(define (vector-index) (vector-ref vect-a 120))

;; String Index Test
(define string-a)
(set! string-a (make-string 128 #\X ))
(define (string-index) (string-ref string-a 120))

;; The good old Prime Number Sieve Test (Test on only 1 iteration)
(define (sieve)
        (letrec ((count 0)              ;; number of primes found
                 (size  7000)           ;; size of sieve array
                 (flags (make-vector (add1 size) 0))
                )
                (do ((i 0 (add1 i)))    ;; scan array from start
                    ((> i size) count)  ;; to finish and return primes found
                    (if (zero? (vector-ref flags i))
                       (let ((prime (+ i i 3)))
                            (do ((k (+ i prime) (+ k prime)))
                                ((> k size) (set! count (add1 count)))
                                (vector-set! flags k 1)
                            )           ;; reset non-prime flags
                       )
                    )
                )
        )
)

;; BYTE Calculation Test (Time only 1 iteration, looping is done internally)
(define (calc)
        (do ((a 2.71828)                ;; setup parameters
             (b 3.14159)
             (c 1.0)
             (i 1 (add1 i))
            )
            ((=? i 5000) (- c 1))       ;; exit when end of test with error
            (set! c (* c a))            ;; perform calculations
            (set! c (* c b))
            (set! c (/ c a))
            (set! c (/ c b))
        )
)

;; End of BYTE TI Scheme Benchmark Source

"BYSO Lisp Benchmark    1-4-86  WGW"
"Test Loop"
(defun loop-test (fn limit)
  (do (( i 1 ( + i 1 )))
      ((= i limit))
      (fn) ) )
(defun dummy ())

"CONS Test"
(setq cons-a nil)
(defun cons-test () (cons cons-a cons-a))

"Integer Addition Test"
(setq add-a 1  add-b 2)
(defun add-test () (+ add-a add-b))

"Integer Multiplication Test"
(setq multiply-a 1   multiply-b 2)
(defun multiply-test () (* multiply-a multiply-b))

"Assignment Test"
(setq assign-a '(1 2 3))
(defun assign-test () (setq assign-a assign-a))
```

continued

March

```
"List Indexing Test"
(setq list-index-list '())
(do ((i 1 (+ i 1)))
    ((= i 128))
    (setq list-index-list (cons i list-index-list)) )
(defun list-index () (nth 120 list-index-list))

"Vector Index Test"
(setq vector-test-array (array 'sexpr 128))
(defun vector-index () (aref vector-test-array 120))

"String Index Test"
(setq string-test-array (array 'char 128))
(defun string-index () (aref string-test-array 120))

"Write test creates a new file and writes 64 kbytes to it."
( defun write-test ()
          ( do-write-test ( open 'b:test )
                          512
                          ( array 'char 128 )
          )
)

( defun do-write-test ( file records buffer )
          ( do ()
               (( zerop ( setq records ( - records 1 ))) ( close file ))
               ( princ buffer file )
          )
)
; Waltz Lisp Benchmark         1-4-86 WGW
;
; Test Loop
(def loop-test (lambda (fn limit)
                  (do ((i 1 ( + i 1 )))
                      ((equal i limit))
                      (fn) ) ))
(def dummy (lambda ()))

; CONS Test
(setq cons-a nil)
(def cons-test (lambda () (cons cons-a cons-a)))

; Integer Addition Test
(setq add-a 1)
(setq add-b 2)
(def add-test (lambda () (+ add-a add-b)))

; Integer Multiplication Test
(setq multiply-a 1)
(setq multiply-b 2)
(def multiply-test (lambda () (* multiply-a multiply-b)))

; Assignment Test
(setq assign-a '(1 2 3))
(def assign-test (lambda () (setq assign-a assign-a)))

; List Indexing Test
(setq list-index-list '())
(do ((i 0 (+ i 1)))
    ((equal i 128))
    (setq list-index-list (cons i list-index-list)) )
(def list-index (lambda () (nth 120 list-index-list)))

; Vector Index Test  (Arrays Not Supported)

; String Index Test
(setq string-test-array "" )
(do ((i 0 (+ i 1)))
    ((equal i 128))
    (setq string-test-array (cat "1" string-test-array)) )
(def string-index (lambda () (substring string-test-array 120 120)))
```

```
; Write test creates a new file and writes 64 kbytes to it.
(def write-test (lambda ()
                     ( do-write-test ( outfile "b:test" )
                                     512
                                     string-test-array ) ))

(def do-write-test (lambda (file records buffer)
          ( do ()
               (( zerop ( setq records ( - records 1 ))) ( close file ))
               ( princ buffer file ) ) ))

;; Golden Common Lisp Benchmark    1-4-86  WGW
;; Test Loop
(defun loop-test (fn limit)
  (do (( i 1 ( + i 1 )))
      ((= i limit))
      (apply fn nil) ) )
(defun dummy () )

;; CONS Test
(setq cons-a nil)
(defun cons-test () (cons cons-a cons-a))

;; Integer Addition Test
(setq add-a 1  add-b 2)
(defun add-test () (+ add-a add-b))

;; Integer Multiplication Test
(setq multiply-a 1   multiply-b 2)
(defun multiply-test () (* multiply-a multiply-b))

;; Floating Point Addition Test
(setq fp-add-a 1.2  fp-add-b 234324.3)
(defun fp-add-test () (+ fp-add-a fp-add-b))

;; Floating Point Multiplication Test
(setq fp-multiply-a 1.2   fp-multiply-b 234324.3)
(defun fp-multiply-test () (* fp-multiply-a fp-multiply-b))

;; Assignment Test
(setq assign-a '(1 2 3))
(defun assign-test () (setq assign-a assign-a))

;; List Indexing Test
(setq list-index-list '())
(do ((i 1 (+ i 1)))
    ((= i 128))
    (setq list-index-list (cons i list-index-list)) )
(defun list-index () (nth 120 list-index-list))

;; Vector Index Test
(setq vector-test-array (make-array 128 :initial-element nil))
(defun vector-index () (aref vector-test-array 120))

;; String Index Test
(setq string-test-array
  (make-array 128 :element-type 'string-char :initial-element 32))
(defun string-index () (aref string-test-array 120))

"Write test creates a new file and writes 64 kbytes to it."
(defun write-test ()
         (do-write-test (open "b:test" :direction ':output)
                        512
                        (make-array 128 :element-type 'string-char)
         )
)

( defun do-write-test ( file records buffer )
          ( do ()
               (( zerop ( setq records ( - records 1 ))) ( close file ))
               ( princ buffer file )
          )
)
```

IPLIST.C
Contributed by Benjamin M. Dawson
"An Introduction to Image Processing Algorithms, March 1987, page 169

listing 1

```
/*  Point process area starting at x,y = 0,0 and of size *
 *  XSIZE,YSIZE.                                          */
  for (y = 0 ; y < YSIZE ; y++) {
        for (x = 0 ; x < XSIZE ; x++) {
                write_pixel(x,y,  pfun(read_pixel(x,y),x,y) );
        }
  }
```

listing 2

```
/* Use long values as sum could be over 16 bits */
  long h[256];
/* zero histogram array */
  for (i = 0 ; i < 256 ; i++) h[i] = 0L;
/* Scan area and count pixel values */
  for (y = 0 ; y < YSIZE ; y++) {
        for (x = 0 ; x < XSIZE ; x++) {
                h[read_pixel(x,y)] = h[read_pixel(x,y)] + 1L;
        }
  }
```

listing 3

```
  long h[256];

/* Histogram area, result into array h */
  histogram(x,y,dx,dy,h);                /* SIMPP routine */
/* Find the low and high bins based on minimum count of 30 */
  clip_histo(h,30,&low_bin,&high_bin);  /* SIMPP routine */
/* Compute the factor for stretching the in between values */
  step = 256.0/(double)(high_bin-low_bin+1);  /* step delta */
  step_value = 0.0;                      /* Step value */
/* Form a translation table (LUT), tran[] for enhancing
   contrast */
/* Values below low_bin are set to minimum pixel value */
  for (i = 0 ; i < low_bin ; i++) tran[i] = 0;
/* Values between low_bin and high_bin are stretched to range
   from 0 to 255 */
  for (i = low_bin ; i <= high_bin ; i++) {
        tran[i] = step_value;
        step_value += step;
  }
/* Values above high_bin are set to maximum pixel value */
  for (i = high_bin+1 ; i < 256 ; i++) tran[i] = 255;
/* Now point process area using the translation table, tran[] */
  while (dy--) {
        for (i = x; i < x + dx; i++) {
                write_pixel(i,y,  tran[read_pixel(i,y)] );
        }
  }
```

listing 4

```
/* Change the output LUTs to display the pixel values *
 * ranging from v_begin to v_end in red.               */
LUT_highlight(v_begin,v_end)
{
  int i;

/* Set output tables to "linear".  This will display
   the image in normal, monochrome fashion */
  for (i = 0 ; i < 256 ; i++) {
        write_LUT(RED,i,i);
        write_LUT(GREEN,i,i);
        write_LUT(BLUE,i,i);
  }
/* Set the desired range so that ONLY red is displayed */
  for (i = v_begin ; i <= v_end ; i++) {
        write_LUT(RED,i,255);   /* Full red */
        write_LUT(GREEN,i,0);   /* No green */
        write_LUT(BLUE,i,0);    /* No blue */
  }
```

listing 5

```
/* Set up kernel for "sharpening" (high-frequency boosting)
   the image */
  static int kernel[9] = {-1,-1,-1,
                          -1, 9,-1,
                          -1,-1,-1,};

/* Increment starting position and decrement image size to accommodate the
   convolution edge effects */
  x++; y++; dx--; dy--;
/* Set up address offsets for the output */
  xx = 0; yy = 0;
/* Scan through source image, output to destination */
  for (i = y ; i < y+dy ; i++) {
     xx = 0;                     /* Reset x output index */
     for (j = x ; j < x+dx ; j++) {
        sum = 0;                 /* Zero convolution sum */
        k_pointer = kernel;      /* Pointer to kernel values */
/* Inner loop to do convolution (correlation!) */
        for ( n = -1 ; n <= 1 ; n++) {
          for (m = -1 ; m <= 1 ; m++)
                sum = sum + read_pixel(j+m,i+n)*(*k_pointer++);
         }
/* Output processing */
        if (sum < 0) sum = 0;
        write_pixel(x_out + xx, y_out + yy, sum);
        xx++;              /* Increment output X address offset */
    } yy++;                /* Increment output Y address offset */
  }
```

listing 6

```
/* Variables used in labeling */
  static int count;
  static int newval = 1;

/* Search image area for target values == 255 */
  for (y = 0 ; y < YSIZE ; y++) {
    for (x = 0 ; x < XSIZE ; x++) {
/* If we find a target value, recursively label
   the connected pixels with a new value (newval) */
        if (read_pixel(x,y) == 255) {
                count = 0;       /* Zero pixel count */
                recursive_label(x,y);
                newval ++j
        }
    }
  }

  recursive_label(x,y)
  {
        write_pixel(x,y,newval);        /* Replace with newval */
        count++;                        /* Increment count */

/* Recurse left */
        x--;
        if (read_pixel(x,y) == 255) recursive_label(x,y);
/* Recurse right */
        x += 2;
```

continued

```
        if (read_pixel(x,y) == 255) recursive_label(x,y);
        x--;
/* Recurse up (remember: video coordinates!) */
        y--;
        if (read_pixel(x,y) == 255) recursive_label(x,y);
/* Recurse down */
        y += 2;
        if (read_pixel(x,y) == 255) recursive_label(x,y);
  }
```

```
listing 7

  int xs,ys;     /* Start of source */
  int x,y;       /* Start of destination */
  int dx,dy;     /* Size of destination area */
  double a,b;    /* x,y scale factors */
  xa,ya;         /* x and y addresses for source */

  for (i = 0 ; i < dy ; i++) {
    for (j = 0 ; j < dx ; j++) {
        xa = xs + (int)((double)j/a);   /* x address */
        ya = ys + (int)((double)i/b);   /* y address */
/* Write out new value to destination */
        write_pixel(x+j, y+i, read_pixel(xa,ya));
    }
  }
```

MAKEFILE
Contributed by Benjamin M. Dawson
"An Introduction to Image Processing Algorithms, March 1987, page 169

```
cat makefile
# Make file for SIMPP
# Requires math library (-lm) only for the gaussian burn function.

CFLAGS =

simtest:        simtest.o siminter.o simprim.o simgeo.o simpoint.o \
                simarea.o simutil.o simsubs.o
                cc -o simtest simtest.o siminter.o simprim.o simgeo.o \
                simpoint.o simarea.o simutil.o simsubs.o -lm
%
```

README
Contributed by Benjamin M. Dawson
"An Introduction to Image Processing Algorithms, March 1987, page 169

```
readme = Introductory file for SIMPP

This directory/distribution contains the source code for SIMPP: SImple Image
Processing Package.  It was written by Benj. Dawson to accompany the article
entitled "An Introduction to Image Processing Algorithms" in the March, 1987
edition of BYTE magazine.

All material is Copyright (c) 1987 by Benajmin M. Dawson.

The complete set of SIMPP files includes the following 13 files:

        makefile
        readme          (This file)
        simarea.c
        simgeo.c
        siminter.c
        simpoint.c
        simpp.doc
        simpp.h
        simprim.c
        simsubs.c
        simtest.c
        simutil.c

See "simpp.doc" for details on use.  See the BYTE article for details on
the algorithms. The listings from the article are contained in the file:

        ipalg.doc
```

SIMAREA.C
Contributed by Benjamin M. Dawson
"An Introduction to Image Processing Algorithms, March 1987, page 169

```
/*****************************************\
*                                         *
*   simarea.c = Area operations for SIMPP:  *
*   Simple IMage Processing Package.        *
*   Copyright (c) 1987, Benjamin M. Dawson  *
*      Edit Version: 1.1 : Jan-29-87        *
*                                         *
\*****************************************/

#include "simpp.h"

extern char *malloc();

/* convolve = Convolve the image area starting at x,y and of size dx,dy
 * with the kernel of size m,n.  Scale (divide) the output by scale, and
 * change the sign of the output values according to the output flag.
 */
int convolve(x,y,dx,dy,m,n,kernel,scale,output)
int x,y;                /* Start of area to convolve */
int dx,dy;              /* Size of area to convolve */
int m,n;                /* Kernel (x,y) size */
int *kernel;            /* Pointer to kernel array */
int scale;              /* Amount to right shift results */
int output;             /* Output flag */
{
        PIXEL *bp[MAX_KERNEL_SIZE];             /* Input pointers */
        PIXEL *ptemp;                           /* Temporary pointer */
        register int i,j;                       /* Loop variables */
        int x_out,y_out;                        /* output x,y index */
        int xx;                                 /* Offset x address */
        int xend;                               /* Reduced x size */
        long sum;                               /* Convolution sum */
        long max_pos;                           /* Maximum + pixel value */
        long maxs_pos;                          /* Maximum signed + value */
        long maxs_neg;                          /* Maximum signed - value */
        int *kp;                                /* Pointer to kernel */

#ifdef CHECK
/* Check source and destination ranges */
        if (check_area(x,y,dx,dy,"<convolve>") == ERROR)
                return(ERROR);
/* Check kernel size */
        if ((m > MAX_KERNEL_SIZE) || (m < 1) ||
            (n > MAX_KERNEL_SIZE) || (n < 1)) {
                printf("<convolution> Kernel size out of range!\n");
                return(ERROR);
            }
/* Special check against kernel size */
        if ((dx < m) || (dy < n)) {
                printf("<convolution> Area too small!\n");
                return(ERROR);
        }
#endif

/* Set up long values for output value checking */
        max_pos = (long)MAXPIX;
        maxs_pos = max_pos/2L;
        maxs_neg = -((long)(PIXEL_SIZE/2));
```

continued

```
/* Allocate line buffers for input */
        for (i = 0 ; i < n ; i++)
                bp[i] = (PIXEL *)malloc(dx*sizeof(PIXEL));

/* Set up addresses and indices */
        x_out = x + (m/2);      /* These factors correct for the convolution
*/
        y_out = y + (n/2);      /*   edge effects (see BYTE article) */
        dy -= (n-1);            /* Reduce area to account for edge effects */
        xend = dx - m;

/* Read first n lines into the input buffers */
        for (i = 0 ; i < n ; i++) read_hline(x,y++,dx,bp[i]);

/* Main convolution loop */
        while (dy--) {                  /* Scan down the image */
          for (xx = 0 ; xx <= xend ; xx++) {    /* Scan across the image */
/* Inner loop */
                sum = 0L;               /* 0 out convolution sum */
                kp = kernel;            /* Set up pointer to kernel */
                for (j = 0 ; j < n ; j++) {

                  ptemp = bp[j] + xx;   /* Point to line area with pixels */
                  for (i = 0 ; i < m ; i++ )
                        sum += ((long)*ptemp++)*((long)*kp++);
                }
/* Scale the output sum quickly with a shift right operation.  Unfortunately,
   a shift right some machines fills 0's rather than the sign bit.  A ifdef
   selects for these unfortunate machines */
#ifdef NO_SIGN_FILL
                if (sum < 0) {
                        sum = -sum;
                        sum >>= (long)scale;
                        sum = -sum;
                }
                else sum >>= (long)scale;
# else
                sum >>= (long)scale;
#endif

/* Output value modified according to output flag */
          switch(output) {
                case SIGNED:            /* Clip values to range from */
                  if (sum < maxs_neg) sum = maxs_neg;   /* maxs_neg to */
                  if (sum > maxs_pos) sum = maxs_pos;   /* maxs_pos */
                  break;
                case POSITIVE:          /* Positive values only. - set to 0 */
                  if (sum < 0L) sum = 0L;
                  if (sum > max_pos) sum = max_pos;
                  break;
                case NEGATIVE:          /* Negative only. Sign inverted. */
                  if (sum > 0L) sum = 0L;
                  sum = - sum;
                  if (sum > max_pos) sum = max_pos;
                  break;
                case ABSOLUTE:          /* Absolute value */
                  if (sum < 0L) sum = -sum;
                  if (sum > max_pos) sum = max_pos;
                  break;
                 }

          write_pixel(x_out+xx,y_out,(PIXEL)sum);       /* Write out value */
          }
/* Shuffle pointers so that all is in order */
        ptemp = bp[0];
        for (i = 0 ; i < n-1 ; i++)     /* Shift buffer pointers */
                bp[i] = bp[i+1];
        bp[n-1] = ptemp;
/* Replace oldest line with new line, and move down a line */
        read_hline(x,y++,dx,bp[n-1]);
        y_out++;
        }

/* Free buffers */
        for (i = 0 ; i < n ; i++) free(bp[i]);
        return(OK);
}
```

```
/* ================================================================= */

/* label = Label an area.  The pixels in the area must be binary with target
 * value == bin1 and background value of bin0.  The area is scanned for
 * connected groups of pixels (blobs).  If a connected area has more
 * than minpix pixels, it's values are changed to a label.  Labels
 * are chosen sequentially starting at blabel and going to elabel, then
 * the label values repeat.  The binary values, bin0 and bin1, must NOT
 * be part of the label set!!!  Areas less than minpix are "killed" by
 * setting them to bin0.  This improves processing speed.
 */

/* Static variables to save stack space */
static int count = 0;                   /* Count of pixels in area */
static PIXEL oldcolor = 0;              /* Target color */
static PIXEL newcolor = 0;              /* Label color */
static int xleft = 0;                   /* Boundary values */
static int ytop = 0;                    /* For checking recursion area */
static int xright = XSIZE-1;
static int ybottom = YSIZE-1;

int label(x,y,dx,dy,bin0,bin1,minpix,blabel,elabel)
int x,y;                /* Start of area to label */
int dx,dy;              /* Size of area to label */
PIXEL bin0;             /* Binary 0 value */
PIXEL bin1;             /* Binary 1 value */
int minpix;             /* Minimum number of connected pixels for label */
PIXEL blabel;           /* Begining value to label with */
PIXEL elabel;           /* End value to label with */
{
        register int i;
        PIXEL lv;

#ifdef CHECK
/* Check area to scan */
        if (check_area(x,y,dx,dy,"<label>") == ERROR)
                return(ERROR);
/* Check that the specified labels are not the same as binary values */
        if ((blabel == bin0) || (blabel == bin1) ||
            (elabel == bin0) || (elabel == bin1)) {
                printf("<label> Labels cannot == binary values!!\n");
                return(ERROR);
        }
#endif

/* Set up boundary values */
        xleft = x;
        ytop = y;
        xright = x+dx-1;
        ybottom = y+dy-1;
/* Set up label value */
        lv = blabel;
/* Search area */
        while (dy--) {
                for (i = x ; i < dx+x ; i++) {
/* If there is a target pixel, fill it */
                        if (read_pixel(i,y) == bin1) {
                                count = 0;       /* Count of pixels */
                                oldcolor = bin1;/* Target color */
                                newcolor = lv;   /* Color to fill with */
                                fill_horiz(i,y);        /* Fill with value */
                                if (count < minpix) {  /* Erase if < minpix */
                                        oldcolor = lv;
                                        newcolor = bin0;
                                        fill_horiz(i,y);
                                }
                                else {                /* Bump color */
                                        if (++lv > elabel) lv = blabel;
                                }
                        }
                }
                y++;
        }
```

continued

```
        return(OK);
}

/* ================================================================ */

/* Recursion/iteration routines for finding and filling connected areas */

static int xl,xr;

/* Horizontal fill recursion.  Does most of the work... */
static VOID fill_horiz(x,y)
int x,y;
{
/* Is this a hit? */
        if (read_pixel(x,y) == oldcolor) {
/* Change as long a horizontal line as you can.  Keep track of x, dx */
                xr = x;
                while (xr <= xright) {
                        if (read_pixel(xr,y) == oldcolor) {
                                write_pixel(xr++,y,newcolor);
                                count++;
                        }
                        else break;
                }

                xl = x-1;
                while (xl >= xleft) {
                        if (read_pixel(xl,y) == oldcolor) {
                                write_pixel(xl--,y,newcolor);
                                count++;
                        }
                        else break;
                }
        xl++;
        if ((xr-xl) > 0) fill_vert(xl,y,xr-xl);
        }
}

/* vertical fill recursion */
static VOID fill_vert(x,y,dx)
int x,y,dx;
{
        while(dx--) {
/* Boundary check and recurse up */
                if (--y >= ytop) fill_horiz(x,y);
                y++;
/* Boundary check and recurse down (Remember: "Video coordinates") */
                if (++y <= ybottom) fill_horiz(x,y);
                y--;
                x++;
        }
}

/* ================ End of simarea.c ================ */

/* <-- FILE BREAK --> */
```

SIMINTER.C
Contributed by Benjamin M. Dawson
"An Introduction to Image Processing Algorithms, March 1987, page 169

```
cat siminter.c
/******************************************************\
*                                                      *
*   siminter.c = Sample interface routines for SIMPP:  *
*   Simple IMage Processing Package.                   *
*   Copyright (c) 1987, Benjamin M. Dawson             *
*      Edit Version: 1.1 : Jan-29-87                   *
*                                                      *
\******************************************************/

/* NOTE:  THESE ARE DUMMY ROUTINES.  THEY PROVIDE A TEMPLATE FOR WRITING
 * YOUR OWN ROUTINES.  YOU MUST WRITE ROUTINES SPECIFIC TO YOUR IMAGE
 * PROCESSING HARDWARE and FRAME MEMORY to USE SIMPP.
 */
#include <stdio.h>
#include "simpp.h"

/* sim_open = Open and initialize image processing hardware */
int sim_open()
{

        return(OK);
}

/* ================================================================= */

/* sim_close = Close image processing hardware */
int sim_close()
{

        return(OK);
}

/* ================================================================= */

/* Acquire = Acquire a single image into the image memory */
VOID acquire()
{

}

/* ================================================================= */

/* read_pixel = Read a single pixel from image memory location x,y */
PIXEL read_pixel(x,y)
int x,y;
{
        return((PIXEL)0);       /* Change to return pixel value ! */
}

/* ================================================================= */

/* write_pixel = Write a single pixel value to image memory location x,y */
VOID write_pixel(x,y,z)
int x,y;
PIXEL z;
{

}

/* ================================================================= */
/* write_LUT = Set a LUT location, loc, to value value in the LUT specified
 * by color (RED, GREEN, or BLUE.  Note:  If you don't have LUTS, it is best
 * to leave this as it is -- a dummy routine.
 */
VOID write_LUT(color,loc,value)
int color,loc;
PIXEL value;
{
#ifdef LUTS

#endif
}

/* ================ End of siminter.c ================ */

/* <-- FILE BREAK --> */
%
```

continued

SIMPOINT.C
Contributed by Benjamin M. Dawson
"An Introduction to Image Processing Algorithms, March 1987, page 169

```
cat simpoint.c
/**********************************************\
*                                              *
*   simpoint.c = Point operations for SIMPP:   *
*   Simple IMage Processing Package.           *
*   Copyright (c) 1987, Benjamin M. Dawson     *
*      Edit Version: 1.1 : Jan-29-87           *
*                                              *
\**********************************************/

#include "simpp.h"

extern char *malloc();

/* ptransform = transform the image according to a function table (LUT) */
int ptransform(x,y,dx,dy,table)
int x,y;                        /* Start of area to transform */
int dx,dy;                      /* Size of area to transform */
PIXEL *table;                   /* Transformation table */
{
        register int i;

#ifdef CHECK
        if (check_area(x,y,dx,dy,"<ptransform>") == ERROR)
                return(ERROR);
#endif
        while (dy--) {
                for (i = x ; i < dx+x ; i++)
                        write_pixel(i, y, table[read_pixel(i,y)]);
                y++;
        }
        return(OK);
}

/* ================================================================= */

/* histogram = histogram the pixel values in the area starting at x,y
 * and of size dx,dy.  The histogram is returned in the long array h.
 */
int histogram(x,y,dx,dy,h)
int x,y;                /* Start of area to histogram */
int dx,dy;              /* Size of area to histogram */
long *h;                /* Array of histogram values */
{
        register int i;
        long *ph;

#ifdef CHECK
        if (check_area(x,y,dx,dy,"<histogram>") == ERROR)
                return(ERROR);
#endif

        ph = h;                 /* Pointer to histogram array */

/* Clear the histogram array */
        for (i = 0 ; i < PIXEL_SIZE ; i++) *ph++ = 0L;
/* Compute the histogram */
        while (dy--) {
                for (i = x ; i < x + dx ; i++)
                        h[read_pixel(i,y)] += 1L;
                y++;
        }

        return(OK);
}

/* ================================================================= */

/* clip_histo = Measure the histogram, h, and return minbin, the first
 * bin above threshold thresh going from bin 0 upwards, and maxbin, the
 * first bin above thrshold thresh going from bin MAXPIX downwards
 */
int clip_histo(h,thresh,minbin,maxbin)
long *h;                        /* Pointer to histogram array */
int thresh;                     /* Threshold for histogram counts */
int *minbin,*maxbin;            /* Returned minimum and maximum bins */
{
        register int i;
        register m;

/* Go from the bottom bin up, looking for a bin above threshold */
        for (i = 0; i < PIXEL_SIZE ; i++) {
                if (h[i] > (long)thresh) break;
        }
        m = *minbin = i;

/* Go from the top bin down, looking for a bin above threshold */
        for (i = (int)MAXPIX ; i > m ; i--) {
                if (h[i] > (long)thresh) break;
        }
        *maxbin = i;

#ifdef CHECK
        if (*maxbin == *minbin) {
    printf("<clip_histo> Histogram has only 1 or no bins above threshold!\n");
                return(ERROR);
        }
#endif

        return(OK);
}

/* ================================================================ */

/* plot_histo = Plot the histogram, h, in the image area starting at
 * x,y and of size dx,dy.  The histogram is scalled to fit into this
 * area.  It is plotted with pixel value (intensity) z.
 */
int plot_histo(x,y,dx,dy,h,z)
int x,y;                /* Start of area to plot in */
int dx,dy;              /* Size of to plot in */
long h[];               /* Histogram array */
PIXEL z;                /* Inensity value to use for plot */
{
        PIXEL *bp, *buf;
        register int i;
        int length;
        long maxval;
        double xsf,ysf; /* X and Y scale factors */
        double xf;

/* Find the maximum histogram value */
        maxval = 0L;
        for (i = 0 ; i < PIXEL_SIZE ; i++) {
                if (h[i] > maxval) maxval = h[i];
        }

/* Best to include this check to prevent division by 0 */
#ifdef CHECK
        if (maxval == 0L) {
                printf("<plot_histo> No values in histogram!\n");
                return(ERROR);
        }
        if (PIXEL_SIZE == 0) {
                printf("<plot_histo> PIXEL_SIZE == 0!\n");
                return(ERROR);
        }
#endif

/* Check plotting area */
#ifdef CHECK
        if (check_area(x,y,dx,dy,"<plot_histo>") == ERROR)
                return(ERROR);
#endif
```

continued

```
/* Compute scale factors */
        xsf = (double)dx/(double)PIXEL_SIZE;
        ysf = (double)dy/(double)maxval;

/* Allocate a buffer for drawing */
        bp = buf = (PIXEL *)malloc(dy*sizeof(PIXEL));

/* Fill it with the drawing color */
        for (i = 0 ; i < dy ; i++) *bp++ = z;

/* Draw histogram.  REMEMBER: Video coordinates! (y increases DOWN) */
        xf = (double)x;
        for (i = 0 ; i < PIXEL_SIZE ; i++) {
                length = (int)((double)h[i]*ysf);
                write_vline((int)xf,y+(dy-length),length,buf);
                xf += xsf;
        }

                free(buf);
                return(OK);
        }

        /* ================ End of simpoint.c ================ */

        /* <-- FILE BREAK --> */
        %
```

SIMPP.DOC
Contributed by Benjamin M. Dawson
"An Introduction to Image Processing Algorithms, March 1987, page 169

```
/*************************************************************************

  |=================================================================|
  |             SIMPP = Simple IMage Processing Package.            |
  |                                                                 |
  |              Copyright (c) 1987, Benjamin M. Dawson             |
  |                      Edit 1.2 : Jan. 30, 1987                   |
  | This package may be freely copied, modified, and used for non-  |
  | commercial purposes.  No guarantee is made that this code is    |
  | correct or suitable for any purpose.  This notice, including the|
  | copyright notice, must appear in all copies and modifications.  |
  |=================================================================|

I. Introduction

SIMPP (Simple IMage Processing Package) is a model image processing package
that demonstrate some important and basic algorithms in image processing.

It is written in "standard" (K&R) C and has been compiled and tested on:
  (1) an IBM Personal Computer/AT using the Computer Innovations Inc. C
     compiler (Big model) and an Imaging Technology Inc. Series 100 frame
     memory.
  (2) A DEC VAX 11/750 computer using the Berkeley Unix (4.2) C compiler
     an Adage 3000 (Ikonas) image processor.

Please see my article in the March, 1987 edition of BYTE magazine ("An
Introduction to Image Processing Algorithms") for details on the package
and algorithms.


II. Hardware Requirements

SIMPP assumes that you have simple image processing hardware on your
computer that can acquire, store, access, and display images with 8 bits
of grey-level intensity.  In particular, this hardware must be able to:

1.  Acquire and store a single picture (a "frame") from an image source
(e.g. TV camera, disk, etc.), with an intensity resolution of 8 bits.  This
will give a pixel values ranging from 0 to 255.  Intensities above 255
are clipped to 255, and intensities below 0 are clipped to 0.  These data
are put into the "image memory".
```

2. Access (read and write) this image memory on a pixel-by-pixel basis, as if the picture values were stored in a large matrix of XSIZE columns by YSIZE rows. I suggest a minimum size of YSIZE = 256 rows and XSIZE = 256 columns.

The pixels are organized by "video" coordinates: x values increase from left to right, and y values increase from top to bottom. Thus coordinate (0,0) is at the top,left of the image and (XSIZE-1,YSIZE-1) is the bottom,right point in the image. The image points must be in normal order rather than in interlace or any other order.

3. Display the pixel values on an RGB or monochrome television monitor or other display device (e.g., EGA, printer, etc.).

If your hardware has transformation tables ("Look-Up Tables" or LUTs) for transforming the output pixel values before they are displayed, the SIMPP package can use these tables. Normally these LUTs map a single (monochrome) pixel value to a red, green, blue triple of values. This allows image memory values to be displayed as arbitrary colors (pseudo-color)

III. Software Setup

In order to use SIMPP, you will need to:
  1. Modify the "simpp.h" header to define your hardware to the software.
  2. Write a set of interface routines that access your hardware.
  3. Deal with porting the software to your machine.
  4. Compile and link the software.

1. Values in the header "simpp.h" specify the hardware you are using to the SIMPP software. Here is an annotated copy of the header, showing items you may want to modify (Set off by !! at the beginning of the line):

```
/*********************************************\
*                                             *
*    simpp.h = Include file for SIMPP         *
*    For Simple IMage Processing Package.     *
*    Copyright (c) 1987, Benjamin M. Dawson.  *
*      Edit Version: 1.1 : Jan-29-87          *
*                                             *
\*********************************************/

!! (1) The VOID definition indicates that no useful value is returned from
!! by the function.  This helps document the function and mollifies some
!! automatic code checkers.
/* Type definitions */
#ifndef VOID                    /* VOID: No useful return from function */
#define VOID
#endif

!! (2) These definitions specify image memory size and structure.
/* Storage definitions.  May need to be changed for your hardware!! */
!!  Each pixel (individual image point) is stored in a 8-bit byte,
!!  even if the hardware acquires fewer bits.  For example, if you
!!  have 6-bit pixels, they still must occupy a byte.  A pixel can
!!  occupy more than a byte (a short, for example), but you might
!!  run out of heap space (internal buffers) on a "small" machine.
!!  Try not to change this item.
#define PIXEL unsigned char     /* Pixel type must be an 8-bit value! */

!!  The size of pixel.  This is for 8-bit values.  If your pixels have
!!  fewer bits, change accordingly.  For example, for 6-bit pixels,
!!  define PIXEL_SIZE to be 64.
#define PIXEL_SIZE 256          /* Size of pixel */

!!  The minimum pixel value.  Leave this at 0, if you can.
#define MINPIX (PIXEL)0         /* Minimum pixel value */

!!  The maximum pixel value is automatically computed.  DON'T CHANGE THIS!
#define MAXPIX (PIXEL)(PIXEL_SIZE-1)    /* Maximum pixel value */

!!  Starting index for the image memory.  Leave these at 0, if you can.
#define XSTART 0                /* Starting image memory X address */
#define YSTART 0                /* Starting image memory Y address */
```

continued

```
!!  Change this to indicate the horizontal size of your image memory.
#define XSIZE 512               /* Horizontal (row) size of image memory */

!!  Change this to indicate the vertical size of your image memory.
#define YSIZE 480               /* Vertical (column) size of image memory */

!!  Automatic definitions.  DON'T CHANGE THESE!
#define XEND XSIZE-1            /* Last horizontal pixel address */
#define YEND YSIZE-1            /* Last vertical pixel address */

!! (3) Define CHECK if you want software checking of arguments ranges.
!! A good idea for debugging!
/* Option switches */
#define CHECK                   /* Define CHECK for bounds checking */

!! (4) These define return values for error reporting.
/* Return values */
#define ERROR -1                /* Error return */
#define OK 0                    /* Return OK */

!! (5) These define the maximum kernel size for the convolution, and values
!! specifying how the output of the convolution will be processed.
/* Convolution switches */
!!  This specifies the maximum kernel size.  Used for checking arguments.
#define MAX_KERNEL_SIZE 8       /* Maximum size of kernel */

!!  These specify how to process the result of the convolution.
#define SIGNED 0                /* Don't change convolution output */
#define POSITIVE 1              /* Output + values only. - set to 0 */
#define NEGATIVE 2              /* Set + values to 0, output - of - values */
#define ABSOLUTE 3              /* Output absolute values */

!! (6) External procedure declarations.  Don't change.
/* External declarations */
extern PIXEL read_pixel();
extern PIXEL read_LUT();

!! (7) Change these values to specify the hardware you are using.
/* CPU and image memory (image processor) specific definitions */
!!  MEMSIZE is the largest possible buffer you can allocate in your
!!  CPU.  This size limits the size of the geometric transformations
!!  and some other operations.  For the IBM AT or PC, with CII Big this
!!  is 2^16-20, as shown below.  Other CPUs and operating systems may
!!  allow a larger value.
#define MEMSIZE 65516L          /* Size of largest buffer for CII Big model */

!!  The output values from the convolution routine are scaled (divided
!!  by using a shift function.  Some machines fill with the sign bit on
!!  a right shift (divide by 2) and others don't.  Define this if your
!!  machine does NOT sign fill on right shift.
#undef NO_SIGN_FILL             /* Define if CPU does NOT fill with sign */
                                /* bits on a right shift (see convolution) */

!!  If your image memory (image processing board) has output LUTs that allow
!!  a pixel to be transformed into a red,green,blue triple of values for
!!  display on a color monitor, then define LUTS to use these LUTs.  If you
!!  define this and don't have LUTs, the only damage is larger code.
#define LUTS                    /* Define LUTS if you have output LUTS */
#ifdef LUTS
!!  These select an output LUT for modification.
#define RED 1                   /* Select RED LUT */
#define GREEN 2                 /* Select GREEN LUT */
#define BLUE 3                  /* Select BLUE LUT */
#endif

/* ================ End of simpp.h ================ */
```

2. You must write a set of "interface" routines that link the SIMPP package with your image memory or image processing hardware. These routines are gathered in the "siminter.c" module.

The interface routines are specified below, but not given in this package, as they will be machine specific. A "dummy" version of "siminter.c" is provided to help you write a version specific to your hardware.

```
Primitives:

int sim_open()
     Opens and initializes the imaging hardware.  Returns ERROR or OK.

int sim_close()
     Closes the imaging hardware.  Returns ERROR or OK.

VOID acquire()
     Acquires one image into the image memory and returns when done.

PIXEL read_pixel(x,y)
 int x,y;
     Returns the value of the pixel in location (x,y) of the image memory.

VOID write_pixel(x,y,z)
 int x,y;
 PIXEL z;
     Writes a new pixel value, z, to location (x,y) in the image memory.

VOID write_LUT(color,loc,value)
 int color,loc;
 PIXEL value;
     Set the location specified by loc in the look-up table specified by
     color to value.  If you don't have (or use) LUTs, this should be a
     dummy routine.


3. Porting the software to your machine.

I have tried to make the SIMPP package as portable as possible, sometimes at
the expense of performance.  Hopefully this will make it easy to port to your
particular compiler, CPU, and image processing hardware.

Some notes:

     -- Differences in image processing hardware and host CPU are indicated
        by #define's in "simpp.h", as noted above.  You might have to add some
        #defines for your hardware and compiler.

     -- You will probably have a lot of trouble if your machine does not
        have an 8-bit byte (e.g. a PDP-8).  Then again, you probably don't
        have a C compiler!

     -- Your C compiler must be reasonably complete.  It if follows the K&R
        standard, you should have no problems.  Data types used include:
                unsigned char
                char
                int             (assumed to be short where necessary)
                long            (cast to long where necessary)
                double

     -- Elements of the "standard" C I/O library used include:
                "stdio.h"
                malloc() and free()
                printf(), fprintf(), and scanf()
                open(), read(), write()
                exit()
        You may have to change these calls to use your compiler's versions.
        For example, under some versions of Whitesmiths' C on the PDP-11,
        printf() becomes putfmt(), and the %d field specifier becomes %i.

  malloc:
        It is assumed that malloc() takes an argument of type: unsigned int.
        If your C library requires this argument to be a long and your type
        int is not equal to a long, then calls to malloc must be changed.

        If you don't have malloc() and free(), then you can change the code
        to use static buffers.  You may not be able to use the geometric
        transforms, as they malloc large buffers.

  open:
        The arguments to open vary from library to library.  This distribution
        shows them as appropriate for Berkeley 4.2 Unix.  You may have to
```

continued

change the READ_ONLY and WRITE_ONLY definitions, and reduce the number of arguments to open() from 3 to 2 (drop the 0777 argument).

exit:

The argument to exit indicates what kind of error is returned to the system. The arguments are defined for Berkeley 4.2 Unix in this distribution. You may have to change them to something appropriate for your system.

-- If your compiler does not use ASCII to encode characters, you may have to modify the matches() routine in "simtest.c"

-- All routine names are different in the first 8 characters. You may have to these and/or internal variable names to meet the requirements of your compiler. If your compiler only has 6 character names, you may have to change the subroutine names.

4. Compiling and linking the software.

This SIMPP distribution (version 1.1 -- January 1987) consists of the following files:

makefile	= A Unix-style file for making the test program.
readme	= A short note as to the contents of the directory.
simarea.c	= Area image processing functions.
simgeo.c	= Geometric image processing functions.
siminter.c	= Model hardware interface routines.
simmeas.c	= Image measurement functions.
simpoint.c	= Point image processing routines.
simpp.doc	= This document.
simpp.h	= Hardware definition header file.
simsubs.c	= Subroutines for test program.
simtest.c	= Test program.
simutil.c	= Utility programs.

The C modules (.c) are compiled in the normal fashion and linked with your main program. The test program ("simtest.c") contains a main() call, so you can link with this for a executable program. The "makefile" can be used or modified to automatically build the software and test program.

If you use the test program (simtest.c), the gaussian burn function (in simsubs.c) requires the calculation of an exponential. This is usually covered by the inclusion of a math library.

Table 1 in the BYTE article contains a list of functions in each module, except for simtest.c and simutil.c.

IV. Testing the software.

A rather extensive test program, "simtest.c" is included. This uses a menu to select operations and also has an automatic test sequence. You may want to use this program as a starting point for your program, and you certainly should use it to see that you have ported and compiled everything correctly.

The test program was used to process some of the images in the BYTE article.

V. Notes

The individual files in this package are separated by the special character sequence:
/* <-- FILE BREAK --> */
This helps separate the files if they are concatenated during distribution.

I am delighted to hear from you by letter or electronic mail about the plusses and problems of SIMPP, and any corrections and additions. I cannot be your telephone consultant -- I'm hard to reach and very busy (who isn't!).

If you wish to use this package in a product, reprint, distribute, or use it in a some commercial way, please contact me about licensing.

Happy image processing!

```
Dr. Ben Dawson
E10-120 M.I.T.                  home:   89 Overbrook Drive.
79 Amherst St.                          Wellesley, MA 02181
Cambridge MA, 02139

Tel. (617) 253-5700
ARPA net: BMD@OZ.AI.MIT.EDU

/* ================ End of simpp.doc ================ */

/* <-- FILE BREAK --> */
```

SIMPP.H
Contributed by Benjamin M. Dawson
"An Introduction to Image Processing Algorithms, March 1987, page 169

```
/*********************************************\
*                                             *
*    simpp.h = Include file for SIMPP:        *
*    For Simple IMage Processing Package.     *
*    Copyright (c) 1987, Benjamin M. Dawson.  *
*      Edit Version: 1.2 : Jan-30-87          *
*                                             *
\*********************************************/

/* Type definitions */
#ifndef VOID                    /* VOID: No useful return from function */
#define VOID
#endif

/* Storage definitions.  May need to be changed for your hardware!! */
#define PIXEL unsigned char     /* Pixel type must be an 8-bit value! */
#define PIXEL_SIZE 256          /* Size of pixel */
#define MINPIX (PIXEL)0         /* Minimum pixel value */
#define MAXPIX (PIXEL)(PIXEL_SIZE-1)    /* Maximum pixel value */
#define XSTART 0                /* Starting image memory X address */
#define YSTART 0                /* Starting image memory Y address */
#define XSIZE 512               /* Horizontal (row) size of image memory */
#define YSIZE 480               /* Vertical (column) size of image memory */
#define XEND XSIZE-1            /* Last horizontal pixel address */
#define YEND YSIZE-1            /* Last vertical pixel address */


/* Option switches */
#define CHECK                   /* Define CHECK for bounds checking */

/* Return values */
#define ERROR -1                /* Error return */
#define OK 0                    /* Return OK */

/* Convolution switches */
#define MAX_KERNEL_SIZE 8       /* Maximum size of kernel */
#define SIGNED 0                /* Don't change convolution output */
#define POSITIVE 1              /* Output + values only. - set to 0 */
#define NEGATIVE 2              /* Set + values to 0, output - of - values */
#define ABSOLUTE 3              /* Output absolute values */

/* External declarations */
extern PIXEL read_pixel();
extern PIXEL read_LUT();

/* CPU and image memory (image processor) specific definitions */
#define MEMSIZE 65516L          /* Size of largest buffer for CII Big model */
#undef NO_SIGN_FILL             /* Define if CPU does NOT fill with */
                                /* bits on a right shift (see convolution) */
#define LUTS                    /* Define LUTS if you have output LUTS */
#ifdef LUTS
#define RED 1                   /* Select RED LUT */
#define GREEN 2                 /* Select GREEN LUT */
```

continued

```
#define BLUE 3                     /* Select BLUE LUT */
#endif

/* ================ End of simpp.h ================ */

/* <-- FILE BREAK --> */
```

SIMPRIM.C
Contributed by Benjamin M. Dawson
"An Introduction to Image Processing Algorithms, March 1987, page 169

```
cat simprin.cmic
/*********************************************\
*                                             *
*   simprim.c = Image processing primitives.  *
*   For Simple IMage Processing Package.      *
*   Copyright (c) 1987, Benjamin M. Dawson    *
*      Edit Version: 1.1 : Jan-29-87          *
*                                             *
\*********************************************/

#include "simpp.h"

extern char *malloc();


/* read_hline = Read horizontal line of pixels, starting at x,y and of
 * length n into the buffer pointed to by bp
 */
int read_hline(x,y,n,bp)
int x,y;                          /* Screen starting location */
register int n;                   /* Number of pixels to read */
register PIXEL *bp;               /* Pointer to pixel buffer */
{

#ifdef CHECK
        if ((x < XSTART) || (x > XEND)) {
                printf("<read_hline> X address out of range!\n");
                return(ERROR);
        }
        if ((y < YSTART) || (y > YEND)) {
                printf("<read_hline> Y address out of range !\n");
                return(ERROR);
        }
        if ((n < 0) || (n > XSIZE-x)) {
                printf("<read_hline> Wrong number of pixels!\n");
                return(ERROR);
        }
#endif

        while (n--) *bp++ = read_pixel(x++,y);
        return(OK);
}

/* ================================================================ */

/* write_hline = Write a buffer, bp into a horizontal line of image memory
 * memory pixels, starting at x,y and of length n.
 */

write_hline(x,y,n,bp)
int x,y;                          /* Screen starting location */
register int n;                   /* Number of pixels to write */
register PIXEL *bp;               /* Pointer to pixel buffer */
{

#ifdef CHECK
        if ((x < XSTART) || (x > XEND)) {
                printf("<write_hline> X address out of range!\n");
                return(ERROR);
        }
        if ((y < YSTART) || (y > YEND)) {
                printf("<write_hline> Y address out of range!\n");
                return(ERROR);
        }
        if ((n < 0) || (n > XSIZE-x)) {
                printf("<write_hline> Wrong number of pixels!\n");
                return(ERROR);
        }
#endif

        while (n--) write_pixel(x++,y,*bp++);
        return(OK);
}

/* ================================================================ */

/* read_vline = Read vertical line of pixels starting at location x,y
 * and of length n into buffer bp
 */
int read_vline(x,y,n,bp)
int x,y;                        /* Screen starting location */
register int n;                 /* Number of pixels to read */
register PIXEL *bp;             /* Pointer to pixel buffer */
{

#ifdef CHECK
        if ((x < XSTART) || (x > XEND)) {
                printf("<read_vline> X address out of range!\n");
                return(ERROR);
        }
        if ((y < YSTART) || (y > YEND)) {
                printf("<read_vline> Y address out of range!\n");
                return(ERROR);
        }
        if ((n < 0) || (n > YSIZE-y)) {
                printf("<read_vline> Wrong number of pixels!\n");
                return(ERROR);
        }
#endif

        while (n--) *bp++ = read_pixel(x,y++);
        return(OK);
}

/*=================================================================*/

/* write_vline = Write a vertical line of pixels from buffer bp into
 * frame memory starting at location x,y and of lenght n.
 */
write_vline(x,y,n,bp)
int x,y;                        /* Screen starting location */
register int n;                 /* Number of pixels to write */
register PIXEL *bp;             /* Pointer to pixel buffer */
{

#ifdef CHECK
        if ((x < XSTART) || (x > XEND)) {
                printf("<write_vline> X address out of range!\n");
                return(ERROR);
        }
        if ((y < YSTART) || (y > YEND)) {
                printf("<write_vline> Y address out of range!\n");
                return(ERROR);
        }
        if ((n < 0) || (n > YSIZE-y)) {
                printf("<write_vline> Wrong number of pixels!\n");
                return(ERROR);
        }
#endif

        while (n--) write_pixel(x,y++,*bp++);
        return(OK);
}

/* ================================================================ */
```

continued

```
/* read_area = Read an area of pixels into buffer bp, starting at
 * location x,y and of length n.
 */
int read_area(x,y,dx,dy,bp)
int x,y;                        /* Screen starting location */
int dx,dy;                      /* Size of area */
register PIXEL *bp;             /* Pointer to pixel buffer */
{

#ifdef CHECK
        if (check_area(x,y,dx,dy,"<read_area>") == ERROR)
                return(ERROR);
#endif
        while (dy--) {
                read_hline(x,y++,dx,bp);
                bp += dx;
        }
        return(OK);
}

/* ================================================================== */

/* write_area = Write the contents of buffer bp into the image area
 * starting at x,y and of size dx,dy.
 */
write_area(x,y,dx,dy,bp)
int x,y;                        /* Screen starting location */
int dx,dy;                      /* Size of area */
register PIXEL *bp;             /* Pointer to pixel buffper */
{
#ifdef CHECK
        if (check_area(x,y,dx,dy,"<write_area>") == ERROR)
                return(ERROR);
#endif
        while (dy--) {
                write_hline(x,y++,dx,bp);
                bp += dx;
        }

        return(OK);
}

/* ================================================================== */

/* copy_area = copy the image memory area starting at x,y and of size
 * dx,dy into the area starting at xd,yd and of size dxd,dyd.
 */
copy_area(x,y,dx,dy,xd,yd,dxd,dyd)
int x,y;                /* Start of source */
int dx,dy;              /* Size of copy */
int xd,yd;              /* Destination start */
int dxd,dyd;            /* Destination size */
{
        PIXEL *bp, *buf;

#ifdef CHECK
        if (check_area(x,y,dx,dy,"<copy_area> Source") == ERROR)
                return(ERROR);
        if (check_area(xd,yd,dxd,dyd,"<copy_area> Destination")==ERROR)
                return(ERROR);
#endif

/* Select the smaller of the two area sizes */
        if (dxd < dx) dx = dxd;
        if (dyd < dy) dy = dyd;

/* Check that you won't run off the frame memory */
        if ((yd + dy) > YSIZE) dy = YSIZE - yd;
        if ((xd + dx) > XSIZE) dx = XSIZE - xd;

/* Allocate a buffer */
        bp = buf = (PIXEL *)malloc(dx*sizeof(PIXEL));

/* If the source is above the destination, copy top down */
        if (y >= yd) {
                while(dy--) {
                        read_hline(x,y++,dx,bp);
                        write_hline(xd,yd++,dx,bp);
                }
        }
/* Else if destination is above source, copy bottom up */
        else {
                y += dy-1;  yd += dy-1;
                while(dy--) {
                        read_hline(x,y--,dx,bp);
                        write_hline(x,yd--,dx,bp);
                }
        }

        free(buf);
        return(OK);
}

/* ================== ERROR CHECKING =========================== */

/* check_area = Checks that area starting at x,y and of size dx,dy will fit
 * into the image memory size, as defined by XSTART, YSTART, XSIZE, YSIZE
 * in simpp.h  Check all values before returning.
 */
#ifdef CHECK
int check_area(x,y,dx,dy,s)
int x,y;        /* Start of area */
int dx,dy;      /* Size of area */
char *s;        /* String to prepend to error statements */
{
        int flag;
        flag = OK;

/* Starting X */
        if ((x < XSTART) || (x > XEND)) {
                printf("%s area X address out of range!\n",s);
                flag = ERROR;
        }
/* Starting Y */
        if ((y < YSTART) || (y > YEND)) {
                printf("%s area Y address out of range!\n",s);
                flag = ERROR;
        }
/* X size */
        if ((dx < 0) || (dx > XSIZE-x)) {
                printf("%s area X size out of range!\n",s);
                flag = ERROR;
        }
/* Y size */
        if ((dy < 0) || (dy > YSIZE-y)) {
                printf("%s area Y size out of range!\n",s);
                flag = ERROR;
        }

        return(flag);
}
#endif

/* ================ End of simprim.c ================ */

/* <-- FILE BREAK --> */
%
```

SIMSUBS.C
Contributed by Benjamin M. Dawson
"An Introduction to Image Processing Algorithms, March 1987, page 169

```
cat simsubs.c
/***********************************************************************\
*                                                                       *
*    simsubs.c = Subroutines for simtest.c program for testing SIMPP:  *
*    For Simple IMage Processing Package.                               *
*    Copyright (c) 1987, Benjamin M. Dawson.                            *
*      Edit Version: 1.1 : Jan-29-87                                    *
```

continued

```
 *                                                                      *
\**********************************************************************/

#include <stdio.h>
#include "simpp.h"

extern double exp();                    /* REQUIRES MATH LIBRARY! */
static PIXEL tran[PIXEL_SIZE] = 0;

/* ==================== Additional Point routines =================== */

/* +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
VOID negate(x,y,dx,dy)          /* Negate an area */
int x,y,dx,dy;                  /* Area to negate */
{
        register int i;

        for (i = 0 ; i < PIXEL_SIZE ; i++)      /* Set up transformation */
                tran[i] = MAXPIX - (PIXEL)i;

        ptransform(x,y,dx,dy,tran);             /* Apply it */
}

/* +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
VOID brighten(x,y,dx,dy,val)    /* brighten an area.  Dim if val < 0 */
int x,y,dx,dy;                  /* area to brighten */
int val;                        /* Value to add to area */
{
        register int i;

        if (val > 0) {  /* BRIGHTEN!! */
          for (i = 0 ; i < PIXEL_SIZE ; i++) {  /* Set up transform */
                if ((i+val) > (unsigned int)MAXPIX) tran[i] = MAXPIX;
                else tran[i] = (PIXEL)(i+val);
          }
        }
        else {          /* DIM !! */
          for (i = 0 ; i < PIXEL_SIZE ; i++) {  /* Set up transform */
                if ((i+val) < (int)MINPIX) tran[i] = MINPIX;
                else tran[i] = (PIXEL)(i+val);
          }
        }

        ptransform(x,y,dx,dy,tran);             /* Apply it */
}

/* +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/* "Burning" is a photographic technique to enhance the exposure and
 * hence the contrast in selective image areas.  This is done by moving
 * a mask with a cutout (usually a circle) around the area while it
 * is being enlarged.  We approximate this procedure using a guassian
 * shaped enhancement of the contrast.
 */
VOID gauss_burn(x,y,dx,dy,k,l,m)        /* Gaussian "Burn" an area */
int x,y,dx,dy;          /* Area to burn */
double k;               /* Expansion (burn) factor */
double l;               /* X and Y space constants */
double m;               /* Offset factor */
{
        register int i;
        double xx, yy, z;

        yy = -(double)dy/2.0;

        while (dy--) {
          xx = -(double)dx/2.0;
          for (i = x ; i < x+dx ; i++) {
                z = (double)read_pixel(i,y)*k*exp(-((xx*xx + yy*yy)/l)) - m;
                if (z < 0.0) write_pixel(i,y,MINPIX);
                else if (z > (double)MAXPIX) write_pixel(i,y,MAXPIX);
                else write_pixel(i,y, (PIXEL)z);
                xx += 1.0;
          }
          yy += 1.0;
```

```
                y++;
        }
}

/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
VOID binarize(x,y,dx,dy,threshold)              /* Binarize area */
int x,y,dx,dy;          /* Area to binarize */
int threshold;          /* Threshold value */
{
        register int i;

        for (i = 0 ; i <= threshold ; i++) tran[i] = MINPIX;
        for (i = threshold ; i < PIXEL_SIZE ; i++) tran[i] = MAXPIX;
        ptransform(x,y,dx,dy,tran);
}

/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
VOID cstretch(x,y,dx,dy,clip)           /* Stretch contrast in area */
int x,y,dx,dy;                          /* Area to enhance */
int clip;                               /* Minimum bin count */
{
        register int i;
        int low_bin,high_bin;           /* Histogram high, low bins */
        double step,step_value;         /* For histogram equalization */
        long h[PIXEL_SIZE];             /* Histogram array */

        histogram(x,y,dx,dy,h);         /* Histogram area */

        clip_histo(h,clip,&low_bin,&high_bin);  /* get min,max histo bins */
        printf("cstretch: Minumum bin = %d, maximum = %d\n",low_bin,high_bin);

        step = (double)PIXEL_SIZE/(double)(high_bin-low_bin+1);/* step size */
        step_value = 0.0;               /* Step value */
/* Values below low_bin are set to minimum pixel value */
        for (i = 0 ; i < low_bin ; i++) tran[i] = MINPIX;
/* Values between low_bin and high_bin are stretched to range from MINPIX to
   MAXPIX */
        for (i = low_bin ; i <= high_bin ; i++) {
                tran[i] = (PIXEL)step_value;
                step_value += step;
        }
/* Values above high_bin are set to MAXPIX */
        for (i = high_bin+1 ; i < PIXEL_SIZE ; i++) tran[i] = MAXPIX;

/* Transform area by stretched values */
        ptransform(x,y,dx,dy,tran);
}

/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
VOID print_histogram(h)         /* Print histogram h */
long *h;
{
        register int i,count;

        printf("Histogram values:\n");
        count = 0;
        for (i = 0 ; i < PIXEL_SIZE ; i++) {
                printf("%6ld ",*h++);
                if (!((++count)%8)) printf("\n");
        }
}


/* ======== Subroutines for setting up LUTS, if you have them ========= */

VOID lin_luts()         /* Set LUTS to linear (grey-scale) */
{
#ifdef LUTS
        register int i;

        printf("-- Liniarize output LUTS --\n");
        for (i = 0 ; i < PIXEL_SIZE ; i++) {
                write_LUT(RED,i,(PIXEL)i);
                write_LUT(GREEN,i,(PIXEL)i);
                write_LUT(BLUE,i,(PIXEL)i);
```

continued

```
        }
#endif
}

/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */

VOID p_spectrum()       /* Set LUTS to a spectrum (pseudo-color) */
{
#ifdef LUTS
        register int i;
        PIXEL j,k;
        int q1,q2,q3,q4;        /* Quartiles of LUT range */

        printf("-- Set output LUTs to a spectrum --\n");

        q1 = PIXEL_SIZE>>2;     /* Divide LUT range by 4 */
        q2 = q1 + q1;           /* Set up quartiles */
        q3 = q2 + q1;
        q4 = PIXEL_SIZE;

/* First quartile of LUT range.  Red increases only */
        for (i = 0 ; i < q1 ; i++) {
                write_LUT(RED,i,(PIXEL)(i<<2)); /* Red ramp up */
                write_LUT(GREEN,i,MINPIX);      /* 0 green */
                write_LUT(BLUE,i,MINPIX);       /* 0 blue */
        }
/* Second quartile of LUT range.  Red decreases, green increases */
        j = MINPIX; k = MAXPIX;
        for (i = q1 ; i < q2 ; i++) {
                write_LUT(RED,i,k);             /* Red decreases */
                k -= 4;
                write_LUT(GREEN,i,j);           /* Green incrases */
                j += 4;
                write_LUT(BLUE,i,MINPIX);       /* 0 blue */
        }
/* Third quartile of LUT range.  Green decreases, Blue increases */
        j = MINPIX; k = MAXPIX;
        for (i = q2 ; i < q3 ; i++) {
                write_LUT(RED,i,MINPIX);        /* 0 red */
                write_LUT(GREEN,i,k);           /* Green decreases */
                k -= 4;
                write_LUT(BLUE,i,j);            /* Blue increases */
                j += 4;
        }
/* Forth quartile of LUT range.  Blue decreases */
        k = MAXPIX;
        for (i = q3 ; i < q4 ; i++) {
                write_LUT(RED,i,MINPIX);        /* 0 red */
                write_LUT(GREEN,i,MINPIX);      /* 0 green */
                write_LUT(BLUE,i,k);
                k -= 4;
        }
#endif
}

/* ================ End of simsubs.c ================ */

/* <-- FILE BREAK --> */
%
```

SIMTEST.C
Contributed by Benjamin M. Dawson
"An Introduction to Image Processing Algorithms, March 1987, page 169

```
/*****************************************\
*                                         *
*   simtest.c = Test program for SIMPP:   *
*   Simple IMage Processing Package.      *
*   Copyright (c) 1987, Benjamin M. Dawson *
*      Edit Version: 1.1 : Jan-29-87      *
*                                         *
\*****************************************/
```

```
#include <stdio.h>
#include "simpp.h"

/* These error flags are returned by the exit() routine.  I have used the
 * values appropriate for UNIX.  These values may not give the desired
 * results under VMS, RT-11, etc., so you may have to change them to
 * values appropriate for your system.
 */
#define ERROR_EXIT      1               /* Signal ERROR exit */
#define OK_EXIT         0               /* Signal no error exit */

/***** Kernels *****/
static int kersh1[]    = {-1, -1, -1,            /* Kernel for sharpening */
                          -1,  9, -1,
                          -1, -1, -1,};

static int kersh2[] =    {-1, -1, -1,            /* Kernel for sharpening */
                          -1, 10, -1,            /* Not as strong as #1 */
                          -1, -1, -1,};          /* Use scale factor of 1 */

static int kersh3[] =    {-1, -1, -1,            /* Kernel for sharpening */
                          -1, 12, -1,            /* Not as strong as #1 or #2 */
                          -1, -1, -1,};          /* Use a scale of 2 */

static int kerhoriz[] = { -1, -1, -1, -1, -1,    /* Kernel for horiz. edges */
                           0,  0,  0,  0,  0,
                           1,  1,  1,  1,  1,};

static int kervert[] = { -1, 0, 1,               /* Kernel for vertical edges */
                         -1, 0, 1,
                         -1, 0, 1,
                         -1, 0, 1,
                         -1, 0, 1,};

static int kerlapla[] = {-1, -1, -1,             /* Kernel for laplacian */
                         -1,  8, -1,
                         -1, -1, -1,};

static int kerblur[] = { 1, 1, 1, 1, 1, 1, 1, 1,
                         1, 1, 1, 1, 1, 1, 1, 1,
                         1, 1, 1, 1, 1, 1, 1, 1,
                         1, 1, 1, 1, 1, 1, 1, 1,
                         1, 1, 1, 1, 1, 1, 1, 1,
                         1, 1, 1, 1, 1, 1, 1, 1,
                         1, 1, 1, 1, 1, 1, 1, 1,
                         1, 1, 1, 1, 1, 1, 1, 1,
                       };

static long h[PIXEL_SIZE] = 0L;                  /* Histogram array */
static char name[80] = 0;                        /* File name */

#define MAX_QUAD 5      /* Maximum quadrant number */

/* Define "quadrants" of image memory, plus "quadrant 0" for the
   entire image memory.  Used as a shorthand for image areas */
#define QUAD0 XSTART,YSTART,XSIZE,YSIZE
#define QUAD1 XSIZE/2,YSTART,XSIZE/2,YSIZE/2
#define QUAD2 XSTART,YSTART,XSIZE/2,YSIZE/2
#define QUAD3 XSTART,YSIZE/2,XSIZE/2,YSIZE/2
#define QUAD4 XSIZE/2,YSIZE/2,XSIZE/2,YSIZE/2
#define QUAD5 XSIZE/4,YSIZE/4,XSIZE/2,YSIZE/2

static int qmap[MAX_QUAD+1][4] ={                /* Quadrant array map */
                          { QUAD0 },             /* Entire screen */
                          { QUAD1 },             /* Upper right */
                          { QUAD2 },             /* Upper left */
                          { QUAD3 },             /* Lower left */
                          { QUAD4 },             /* Lower right */
                          { QUAD5 },             /* Center */
                        };
/* Define macro to use quad map */
#define QUADMAC(i) qmap[i][0],qmap[i][1],qmap[i][2],qmap[i][3]

/* ============================================================ */
```

continued

```
main()
{
        register int i;                     /* General variable */
        int x,y,z;                          /* Location and value variables */
        int out;                            /* Output quadrant */
        char s[20];                         /* Input command string */
        char sharp[5];                      /* Sharpen command string */
        float xs,ys;                        /* Stretch factors */
        float gbk,gbl,gbm;                  /* Gaussian burn factors */

/* Salutations! */
        printf("****** SIMPP test program V1.10 ******\n");
        printf("Copyright (c) 1987, Benjamin M. Dawson\n\n");

/* Open hardware */
        printf("-- Open hardware --\n");
        if (sim_open() == ERROR) {
                printf("<simtest> Can't open hardware!\n");
                exit(ERROR_EXIT);
        }

/* Linearize LUTS, if you have them */
        lin_luts();
        printf("\n");

        for (;;) {

/* Print menu */
        printf("Acquire\t       Binarize         Brighten         Burn\n");
        printf("Blur\t          Clear area\tCopy area   Contrast Stretch\n");
        printf("Grid\t          Histogram        Horizontal Edges\n");
        printf("Label           Laplacian        Linear LUTs\n");
        printf("Negate\t        Plot Histogram   Print Histogram\n");
        printf("Quit            Rotate           Restore image\n");
        printf("Save image      Sharpen          Spectrum\n");
        printf("Stretch\t       Test all         Vertical Edges\n");
        printf("Command>>> ");

        scanf("%s",s);          /* Get command string, with appended NULL */

/* Match command */
        if (matches(s,"acquire",4)) acquire();
        else if (matches(s,"binarize",4)) {
                i = get_quad();
                binarize(QUADMAC(i),get_int("Threshold ="));
        }
        else if (matches(s,"brighten",4)) {
                i = get_quad();
                brighten(QUADMAC(i),get_int("Value to add="));
        }
        else if (matches(s,"blur",4)) {
                i = get_quad();
                convolve(QUADMAC(i),8,8,kerblur,6,POSITIVE);
        }
        else if (matches(s,"burn",4)) {
                i = get_quad();
                printf("Burn factor (real) =");
                scanf("%f",&gbk);
                printf("Space constant (real) =");
                scanf("%f",&gbl);
                printf("Offset (real) =");
                scanf("%f",&gbm);
                gauss_burn(QUADMAC(i),(double)gbk,(double)gbl,(double)gbm);
        }
        else if (matches(s,"clear",4)) {
                i = get_quad();
                clear_area(QUADMAC(i),(PIXEL)get_int("Clear to intensity ="));
        }
        else if (matches(s,"copy",4)) {
                printf("Copy ");
                i = get_quad();
                printf("To ");
                out = get_quad();
                copy_area(QUADMAC(i),QUADMAC(out));
```

```
}
else if (matches(s,"contrast",4)) {
        i = get_quad();
        cstretch(QUADMAC(i),get_int("Minimum bin count ="));
}
else if (matches(s,"grid",4)) {
        x = get_int("X spacing =");
        y = get_int("Y spacing =");
        z = get_int("Intensity =");
        draw_grid(x,y,(PIXEL)z);
}
else if (matches(s,"histogram",4)) {
        i = get_quad();
        histogram(QUADMAC(i),h);
}
else if (matches(s,"horizontal",4)) {
        i = get_quad();
        convolve(QUADMAC(i),5,3,kerhoriz,0,ABSOLUTE);
}
else if (matches(s,"label",4)) {
        i = get_quad();
        x = get_int("Minimum area size =");
        label(QUADMAC(i),MINPIX,MAXPIX,x,
                (PIXEL)1,(PIXEL)(PIXEL_SIZE-2));
}
else if (matches(s,"laplacian",4)) {
        i = get_quad();
        convolve(QUADMAC(i),3,3,kerlapla,0,ABSOLUTE);
}
else if (matches(s,"linear",4)) lin_luts();
else if (matches(s,"negate",4)) {
        i = get_quad();
        negate(QUADMAC(i));
}
else if (matches(s,"plot",4)) {
        i = get_quad();
        plot_histo(QUADMAC(i),h,
                (PIXEL)get_int("Intensity to plot with = "));
}
else if (matches(s,"print",4)) print_histogram(h);
else if (matches(s,"quit",4)) {
        sim_close();
        exit(OK_EXIT);
}
else if (matches(s,"restore",4)) {
        i = get_quad();
        printf("File name:");
        scanf("%s",name);
        read_image(QUADMAC(i),name);
}
else if (matches(s,"rotate",4)) {
        i = get_quad();
        rotate(QUADMAC(i));
}
else if (matches(s,"save",4)) {
        i = get_quad();
        printf("File name:");
        scanf("%s",name);
        save_image(QUADMAC(i),name);
}
else if (matches(s,"sharpen",4)) {
        i = get_quad();
        printf("Degree of sharpening: High, Medium, or Low:");
        scanf("%s",sharp);
        if (matches(sharp,"high",3))
                convolve(QUADMAC(i),3,3,kersh1,0,POSITIVE);
        else if (matches(sharp,"medium",3))
                convolve(QUADMAC(i),3,3,kersh2,1,POSITIVE);
        else if (matches(sharp,"low",3))
                convolve(QUADMAC(i),3,3,kersh3,2,POSITIVE);
}
else if (matches(s,"spectrum",4)) p_spectrum();
else if (matches(s,"stretch",4)) {
        i = get_quad();
```

continued

```
                printf("X stretch factor (real) = ");
                scanf("%f",&xs);
                printf("Y stretch factor (real) = ");
                scanf("%f",&ys);
                stretch(QUADMAC(i),(double)xs,(double)ys);
          }
          else if (matches(s,"test",4)) test_all();
          else if (matches(s,"vertical",4)) {
                i = get_quad();
                convolve(QUADMAC(i),3,5,kervert,0,ABSOLUTE);
          }

          else printf("?? Not a valid command!\n");
          }
}

/* +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
static VOID test_all()          /* Script to test the package */

{
        register int i;
        char c;

/* Point operations */
        printf("\n Test Point operations:\n");
        printf("-- Take a picture --\n"); acquire();
        printf("-- Negate second Quadrant --\n"); negate(QUAD2);
        printf("-- Brighten first Quadrant by 30 --\n"); brighten(QUAD1,30);
        printf("-- Contrast stretch third Quadrant --\n"); cstretch(QUAD3,30);

#ifdef LUTS
        printf("\nTest Output LUTs:\n");
        p_spectrum();           /* Set luts to spectrum */
        printf("Type return (ENTER) to continue:");
        scanf("%c",&c);
        lin_luts();             /* Linearize LUTS */
#endif

/* Area operations */
        printf("\nTest Area operations:\n");

        printf("-- Take a picture --\n"); acquire();
        printf("-- Sharpen image in Quadrant 1 --\n");
        convolve(QUAD1,3,3,kersh1,0,POSITIVE);
        printf("-- Horizontal edges in Quadrant 2 --\n");
        convolve(QUAD2,5,3,kerhoriz,0,ABSOLUTE);
        printf("-- Vertical edges in Quadrant 3 --\n");
        convolve(QUAD3,3,5,kervert,0,ABSOLUTE);
        printf("-- Laplacian of Qandrant 4 --\n");
        convolve(QUAD4,3,3,kerlapla,0,ABSOLUTE);

        printf("Area operations test done.\n");
        printf("Type return (ENTER) to continue:");
        scanf("%c",&c);

/* Geometric operations */
        printf("\nTest Geometric operations:\n");
        printf("-- Take a picture --\n"); acquire();
        printf("-- Draw calibration grid --\n"); draw_grid(10,15,MAXPIX);
        printf("-- Rotate Quadrant 2 --\n"); rotate(QUAD2);
        printf("-- Stretch Quadrant 1 by 2.0 in X --\n");
        stretch(QUAD1,2.0,1.0);
        printf("-- Stretch Quadrant 3 by 2.0 in Y --\n");
        stretch(QUAD3,1.0,2.0);
        printf("-- Stretch Quadrant 4 by 3 in X and Y (ZOOM) --\n");
        stretch(QUAD4,3.0,3.0);

        printf("Geometric operations test done.\n");
        printf("Type return (ENTER) to continue:");
        scanf("%c",&c);

/* Measurement operations */
        printf("\nTest Measurement operations:\n");
        printf("-- Taking a picture --\n"); acquire();
        printf("-- Histogram Quadrant 2 --\n"); histogram(QUAD2,h);
        print_histogram(h);
        printf("-- Plot histogram in Quadrant 1 --\n");
        clear_area(QUAD1,MINPIX);
        plot_histo(QUAD1,h,MAXPIX);

        printf("-- Copy Quad 3 to Quad 4 --\n");
        copy_area(QUAD3,QUAD4);
        printf("-- Binarize Quadrants 3 and 4 --\n");
        binarize(QUAD3,PIXEL_SIZE/2);
        binarize(QUAD4,PIXEL_SIZE/2);

#ifdef LUTS             /* If you have LUTS, use them to color image */
        printf("-- Setting LUTs to show labeled areas --\n");
        p_spectrum();
/* Leave the binary colors (MINPIX and MAXPIX) alone */
        write_LUT(RED,MAXPIX,MAXPIX);
        write_LUT(GREEN,MAXPIX,MAXPIX);
        write_LUT(BLUE,MAXPIX,MAXPIX);
        write_LUT(RED,MINPIX,MINPIX);
        write_LUT(GREEN,MINPIX,MINPIX);
        write_LUT(BLUE,MINPIX,MINPIX);
#endif
        printf("Label binary objects with 20 pixels or more in Quadrant 4\n");
        label(QUAD4,MINPIX,MAXPIX,20,(PIXEL)1,(PIXEL)(PIXEL_SIZE-2));
        printf("Measurement operations test done.\n");

        printf("Type return (ENTER) to continue:");
        scanf("%c",&c);

/* Utility operations */
        printf("\Test Utility operations:\n");
        printf("-- Take a picture --\n"); acquire();

        printf("-- Save Quadrant 2 --\n");
        printf("File name:");
        scanf("%s",name);
        save_image(QUAD2,name);
        printf("-- Restore Quadrant 2 image to Quadrant 4 --\n");
        read_image(QUAD4,name);

        printf("Utility operations test done.\n");
        printf("\n TEST DONE\n");

}

/* ======================= Internal routines ======================= */

/* WARNING:  This assumes an ASCII encoding for your characters!!! */
/* Encode and match two strings, up to string length n or a null.
   Returns 1 if match, 0 elsewise */
static int matches(s1,s2,n)
char *s1,*s2;   /* Strings to match */
int n;          /* Number of characters to use */
{
        register int i,v;

/* If either string is NULL, return no match */
        if ((*s1 == NULL) || (*s2 == NULL)) return(0);

        v = 0;
        for (i = 0 ; i < n ; i++) {
/* Nulls always match after one character */
                if ((*s1 == NULL) || (*s2 == NULL)) return(1);
/* Lower case elements.  REQUIRES ASCII ENCODING!!! */
                if ((*s1 > 0100) && (*s1 <= 0132)) *s1 += 040;
                if ((*s2 > 0100) && (*s2 <= 0132)) *s2 += 040;
                if (*s1++ != *s2++) return(0);
        }
        return(1);
}
/* +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
static int get_quad()           /* Get a quadrant number (0,1,2,3,4) */
{
        int q;
```

continued

```
        for (;;) {
                printf("Quadrant number = ");
                scanf("%d",&q);
                if ((q < 0) || (q > MAX_QUAD))
                  printf("Quadrant number must be 0 (for all) to %d!\n",
                        MAX_QUAD);
                else
                  return(q);
        }
}

/* +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
int get_int(s)          /* Print string and return an integer value */
char *s;
{
        int i;

        printf("%s ",s);
        scanf("%d",&i);
        return(i);
}


/* ================ End of simtest.c ================ */

/* <-- FILE BREAK --> */
```

SIMUTIL.C
Contributed by Benjamin M. Dawson
"An Introduction to Image Processing Algorithms, March 1987, page 169

```
/***********************************************\
*                                               *
*   simutil.c = Utility operations for SIMPP:   *
*   Simple IMage Processing Package.            *
*   Copyright (c) 1987, Benjamin M. Dawson      *
*      Edit Version: 1.2 : Jan-30-87            *
*                                               *
\***********************************************/

#include <stdio.h>
#include "simpp.h"

extern char *malloc();

/* These flags are for open() to indicate read/write ability on a file.
 * The are defined for Berkeley UNIX, but may have to be changed for your
 * system.
 */
#define READ_ONLY  00000
#define WRITE_ONLY 01001

/* clear_area = Clear an area , begining at x,y and of size dx,dy to
 * value z.
 */
int clear_area(x,y,dx,dy,z)
int x,y;                        /* Start of area to set */
int dx,dy;                      /* Size of area to set */
PIXEL z;                        /* Value to set area to */
{
        register int i;
        register PIXEL *bp;
        PIXEL *buf;

#ifdef CHECK
        if (check_area(x,y,dx,dy,"<clear_area>") == ERROR)
                return(ERROR);
#endif
/* Set up a buffer with size of dx */
        buf = bp = (PIXEL *)malloc(dx*sizeof(PIXEL));
/* Set it to all z values */
        for (i = 0 ; i < dx ; i++) *bp++ = z;
```

```
/* Clear lines */
        while (dy--) write_hline(x,y++,dx,buf);

        free(buf);
        return(OK);
}

/* ================================================================ */

/* save_image = Save the image starting at x,y and of size dx,dy into the
 * filewith name name.
 */
int save_image(x,y,dx,dy,name)
int x,y;                        /* Start of area to save */
int dx,dy;                      /* Size of area to save */
char *name;                     /* File name to use for save */
{
        int fd;
        short xx,yy,dxx,dyy;    /* Shorts for header */
        PIXEL *buf, *bp;
        int wrsize;

#ifdef CHECK
        if (check_area(x,y,dx,dy,"<save_image>") == ERROR)
                return(ERROR);
#endif

/* Copy integer values to shorts, so that the header is transportable
   between machines */
        xx = (short)x;  yy = (short)y;
        dxx = (short)dx;  dyy = (short)dy;

/* Open file for writing */
        if ((fd = open(name,WRITE_ONLY,0777)) < 0) {    /* Write only open */
                printf("<save_image> Can't open file %s !\n",name);
                perror("=");
                return(ERROR);
        }

/* Write out header -- xx,yy, dxx,dyy */
        write(fd,&xx,sizeof(short));
        write(fd,&yy,sizeof(short));
        write(fd,&dxx,sizeof(short));
        write(fd,&dyy,sizeof(short));


/* Allocate a buffer */
        bp = buf = (PIXEL *)malloc(dx*sizeof(PIXEL));
/* Write out image data */
        wrsize = dx*sizeof(PIXEL);
        while (dy--) {
                read_hline(x,y++,dx,bp);
                if (write(fd,bp,wrsize) < wrsize) {
                        printf("<save_image> File write error!\n");
                        close(fd);
                        return(ERROR);
                }
        }

        close(fd);              /* Close file */
        free(buf);              /* Free buffer */
        return(OK);
}

/* ================================================================ */

/* read_image = Read an image from disk storage.  If any of the
 * values x,y,dx,dy are < 0, then the corresponding value from the
 * file is used.
 */
int read_image(x,y,dx,dy,name)
int x,y;                        /* Start of area to read */
int dx,dy;                      /* Size of area to read */
char *name;                     /* File name to use for read */
{
```

continued

```
        int fd;
        short xx,yy,dxx,dyy;    /* Shorts for header */
        PIXEL *buf, *bp;
        int rdsize;

/* Open file for reading */
        if ((fd = open(name,READ_ONLY,0777)) < 0) {    /* Open read only */
                printf("<read_image> Can't open file %s !\n",name);
                return(ERROR);
        }

/* Read header -- xx,yy, dxx,dyy */
        read(fd,&xx,sizeof(short));
        read(fd,&yy,sizeof(short));
        read(fd,&dxx,sizeof(short));
        read(fd,&dyy,sizeof(short));

/* Check if you should use these values or the argument values */

        if (x < 0 ) x = (int)xx;
        if (y < 0 ) y = (int)yy;
        if (dx < 0 ) dx = (int)dxx;
        if (dy < 0) dy = (int)dyy;

/* See if you should truncate the size */

        if ((int)dxx < dx) dx = (int)dxx;
        if ((int)dyy < dy) dy = (int)dyy;

/* See if it will run off the screen */

        if ((x + dx) > XSIZE) dx = XSIZE - x;
        if ((y + dy) > YSIZE) dy = YSIZE - y;

/* Allocate a buffer */
        bp = buf = (PIXEL *)malloc(dx*sizeof(PIXEL));

/* Read in image data */
        rdsize = (int)dxx*sizeof(PIXEL);
        while (dy--) {
                if (read(fd,bp,rdsize) < rdsize) {
                        printf("<read_image> Image read error!\n");
                        close(fd);
                        return(ERROR);
                }
                write_hline(x,y++,dx,bp);
        }

        close(fd);              /* Close file */
        free(buf);              /* Free buffer */
        return(OK);
}

/* ================================================================= */

/* draw_grid = Cover the screen with a grid of spacing dx,dy, and value z */
VOID draw_grid(dx,dy,z)
int dx,dy;              /* Grid spacing */
PIXEL z;
{
        register int i;
        register PIXEL *bp;
        PIXEL *buf;

/* Which direction, horizontal or vertical, is longer? */
        if (XSIZE > YSIZE) i = XSIZE;
        else i = YSIZE;
/* Allocate a buffer of this size, and fill it with value z */
        buf = bp = (PIXEL *)malloc(i*sizeof(PIXEL));
        while (i--) *bp++ = z;
/* Draw the lines */
        for (i = 0 ; i < YSIZE ; i += dy) write_hline(0,i,XSIZE,buf);
        for (i = 0 ; i < XSIZE ; i += dx) write_vline(i,0,YSIZE,buf);

        free(buf);
}
```

```
/* ================ End of simutil.c ================ */

/* <-- FILE BREAK --> */
```

SIMGEO.C
Contributed by Benjamin M. Dawson
"An Introduction to Image Processing Algorithms, March 1987, page 169

```
/***********************************************\
*                                               *
*   simgeo.c = Geometric Operations for SIMPP:  *
*   Simple IMage Processing Package.            *
*   Copyright (c) 1987, Benjamin M. Dawson      *
*      Edit Version: 1.2 : Jan-30-87            *
*                                               *
\***********************************************/

#include "simpp.h"

extern char *malloc();

/* rotate = Rotate clockwise by 90 degrees.  Limited to an area of MEMSIZE,
 * and should be a square area.
 */
int rotate(x,y,dx,dy)
register int x,y;       /* Start of area to rotate */
int dx,dy;              /* Size of area to rotate */
{
        register int i;
        unsigned long nl;
        PIXEL *buf, *bp;

#ifdef CHECK
        if (check_area(x,y,dx,dy,"<rotate>") == ERROR)
                return(ERROR);
#endif

/* Compute the number of bytes in the image area to be rotated */
        nl = ((long)dx)*((long)dy)*((long)sizeof(PIXEL));
#ifdef CHECK
        if (nl > MEMSIZE) {
                printf("<rotate> Area too large!\n");
                return(ERROR);
        }
        if (dx != dy) {
                printf("(rotate) Warning: area not square, may write\n");
                printf("     outside of the image memory!\n");
        }
#endif
/* Allocate a large buffer for storage */
        buf = bp = (PIXEL *)malloc(((unsigned int)nl));

/* Read in image */
        if (read_area(x,y,dx,dy,bp) == ERROR) {
                printf("<rotate> Can't read area!\n");
                return(ERROR);
        }

/* Write it out the other way */
        while (dy--) {
                for (i = 0 ; i < dx ; i++) write_pixel(x,y+i,*bp++);
                x++;
        }

        free(buf);
        return(OK);
}

/* =============================================================== */

/* stretch = Enlarge image size by stretching the image by xs,ys.
 * The image is stretched into the same area, starting from the
```

continued

```
 * upper,left corner.  Limited to a MEMSIZE area, and uses interpolation
 * to smooth pixel values.
 */
stretch(x,y,dx,dy,xs,ys)
int x,y;                /* Start of source AND destination */
int dx,dy;              /* Size of Source AND destination */
double xs,ys;           /* X,Y scale factors */
{
        register int i,j;
        int ix,iy;
        unsigned long nl;
        PIXEL *buf, *bp, *p;
        double A,B,C,D,xAB,xCD,xx,yy,V;

#ifdef CHECK
        if (check_area(x,y,dx,dy,"<stretch>") == ERROR)
                return(ERROR);

/* Must stretch -- shrinking will cause indexing problems */

        if ((xs < 1.0) || (ys < 1.0)) {
                printf("<stretch> Scale factors must be > 1.0!\n");
                return(ERROR);
        }
#endif

        nl = ((long)dx)*((long)dy);
#ifdef CHECK
        if (nl > MEMSIZE) {
                printf("<stretch> Area too large!\n");
                return(ERROR);
        }
#endif

        buf = bp = (PIXEL *)malloc((unsigned int)nl*sizeof(PIXEL));
        if (read_area(x,y,dx,dy,bp) == ERROR) return(ERROR);

/* Method:  We compute fractional addresses (xx,yy) that indicate where
 * the source "pixel" would be.  Since this address, in general, will fall
 * between pixels, we approximate the value by bi-linear interpolation.
 * We round the fractional addresses down to integer values to get the
 * address (ix,iy) of nearest real pixel to the top and left of the desired
 * "interpolated pixel".  We then subtract this address from the interpolated
 * pixel address to get the fraction of address between the four pixels
 * surrounding the interpolated pixel (xx = xx - ix, yy = yy -iy).  The
 * address of the nearest top, left pixel (called pixel A) is incremented
 * to get pixel values for the four pixels surrounding the interpolated
 * pixel.  Thus we have a 2 by 2 matrix of values:
 *                      A   B
 *                      C   D
 * To get the interpolated value, we first compute the linear weighted
 * value at distance xx between A and B and at xx between C and D.  These
 * two values (yAB and yCD) are weighted by yy to get the interpolated
 * value.
 */
/* Compute address of the "interpolated pixel" and the nearest real pixel */
        for (i = 1 ; i < dy ; i++) {    /* Don't change first row */
          yy = (double)(i-1)/ys;        /* Compute y address for source */
          iy = (int)yy;                 /* Round to integer */
          yy = yy - (double)iy;         /* Leave only fraction of address */
          bp = buf + iy*dx;             /* Compute pointer to nearest line */
          for (j = 1 ; j < dx ; j++) {  /* Don't change first column */
                xx = (double)(j-1)/xs;  /* Compute x address for source */
                ix = (int)xx;           /* Rount to integer */
                xx = xx - (double)ix;   /* Leave only fraction of address */
 /* Compute the value of the four nearest neighbor pixels */
                p = bp + ix;            /* Pointer to top, left real pixel */
                A = (double)(*p++);     /* Top, left real pixel */
                B = (double)(*p);       /* Top, right real pixel */
                D = (double)(*(p + dx));/* Bottom, right real pixel */
                C = (double)(*(p + dx - 1));    /* Bottom, left real pixel */
/* Compute the linear interpolated values (A to B and C to D) */
                xAB = A + (B - A)*xx;   /* xx of the way between A and B */
                xCD = C + (D - C)*xx;   /* xx of the way between C and D */
/* Compute the (bi)linear interpolated value between xAB and xCD.  This is
 * yy of the way between xAB and xCD.  Round the value by adding 0.5 */
                V = xAB + ((xCD - xAB)*yy) + 0.5;
/* Write out interpolated value */
                write_pixel(x+j,y+i,(PIXEL)V);
             }
        }

        free(buf);
        return(OK);
}

/* ================ End of simgeo.c ================ */

/* <-- FILE BREAK --> */
```

BEST OF BIX FOR THE '386

With the introduction of the 80386 there's a renewed spirit of inquiry bubbling to the surface in the micro field. By way of example, the following pages contain excerpted dialog, questions, answers, pleas and pontifications, as well as a solid body of technical data, on the new Intel chip, how it's being applied by some vendors, and what you can do with it yourself.

We're presenting it here because we feel that this is a natural vehicle for user reaction to one of the latest advances in microprocessor technology. There's probably a greater interest among Listings readers for front line technology on a "how-does-it-work" level than in any other group we have a method of reaching. We're interested in your reactions. Let us know if this supplement to the Supplement proves interesting or useful.

CURRENT OS OPTIONS

os386/vm #2, from fheilbronner, Tue Nov 4 09:14:12 1986.

Boy did this conference come along at the right time. I've got to replace a number of '286 machines running various DOS derivatives and with the workload I'm anticipating a strictly DOS environment just flat won't cut the mustard. I'm in a time bind also so I can't afford to wait for big-boo to come out with their machine. I'm pretty much restricted to DOS delivery systems/environments but for development purposes I'm open for suggestions

. I've pretty much decided to go with one or more Compaq-386's and try and use them in a closely linked net. The trouble is of course getting a 386-OS that is somthing other than either a temporary stopgap OS or a total kludge. I'd like to get a UNIX based OS with DOS running as a task(s). The trouble, as I currently see it is that there are only 286 versions of UNIX available now, and these bad boys aren't able to take advantage of the 386 (other than just speed-wise). I've talked to the folks over at the Santa Cruz Op. about the availablity of a 386-Xenix but they say it will be 2nd-Q/87. In addition SCO is probably going to make 386-Xenix a totally new product so if you've got 286-Xenix upgrading is gonna COST.

I've heard rumblings from distant lands about other multi-tasking OS's for the 386 that run DOS as a task(s), a VM/386 ?, Intel ?, and others. The 'Crux of the biscuit' is 'Whats a mother to do?'

If anyone out there can point me (and I'm sure, a lot others) in the right direction I'd be redundantly appreciatve!

NOT(fheilbronner)

os386/vm #4, from intel, Fri Nov 7 02:58:16 1986.

Today I saw a demo of Softguard's OS VM/386, I was as blown away with it as I was when I saw the VisiCorp's VisiOn demo (my previous company), MacPaint, the Mac's switcher, or VideoWorks for the Mac. VM/386 is an incrediable product, it also you to create multiple "virtual IBM PC". In each Virtual Machine you can run PC-DOS and any PC-DOS applications.

VM/386 multitasks between each Virtual Machine. (The user can even select the time slice he wishes to allocate to each virtual machine). The user switches between virtual machines via a hot key (Sys-Req) which causes a complete screen switch.

I was really impressed about the functionality of the product at this time. They demonstrated Jet, Lotus, Sargon, Topview, and a basic program all running simultaneously!!. In fact since Topview has limited multitasking Topview was doing multitasking under VM/386. The speed was pretty impressive all the forground process were typically running at AT speed. The thing that I liked was being able to do other tasks while waiting for a program to boot from floppy. In one test I made a 13,000 cell lotus spreadsheet, put it in the background, and started up a game program while I was waiting for 1-2-3 to finish copying the cells and when I came back 1-2-3 was Ready, neat.

The OEM sales manager, Steve Williams says that you can see VM/386 at Comdex at the Softguard, Compaq, and probably at the Intel booths. If you are at Comdex make sure to look at this product.

Clif Purkiser 386 Applications

386 VM

os386/vm #5, from patwood, Sat Nov 8 10:28:11 1986.

Besides SCO, Compaq is doing their own Xenix port to the 386 environment. Microport (a UNIX porting company) has a good 286 product and is working on a 386 port of System V Release 2. They have also licensed the Locus product for their 286 machine (allows simultaneous UNIX/DOS executA≈NKKC⊙also known as Simultask on the AT&T PC 6300+).

Does anyone know of Softguard's VM/386 can run multiple 8086 Xenix systems? Given the advent of 386 Xenix, this may sound useless, but it would be interesting.

Pat Wood

os386/vm #6, from patwood, Sat Nov 8 10:29:08 1986. A comment to message 5.

Oh yes, the tel # for Microport is 800-722-8649.

os386/vm #7, from billn, Mon Nov 17 17:49:55 1986. A comment to message 5.

At this point, protected 286 mode applications cannot run under vm/386 because the 386 virtualization is not complete (of 286 code). Later masks for the 386 may be changed, full virtual is promised for the 486 chip and it may be possible to remove the offending instructions from unix earlier. But not now. Maybe soon. For sure RSN. :-) BillN

os386/vm #8, from msokol, Mon Nov 17 21:42:18 1986. A comment to message 7.

However, 8086 based XENIX systems will execute under VM/386. Marc Sokol

os386/vm #9, from jshiell, Tue Nov 18 00:54:15 1986. A comment to message 8.

Note that some protected mode applications could be run under vm/386. That is those that "know" that they are running under VM/386 or are well behaved or are user mode only. Jon Shiell

os386/vm #10, from fheilbronner, Tue Nov 18 11:27:20 1986. A comment to message 7.

Could you explain what you said about 386 virtualization not being complete? Why would the 386 masks have to be changed and who would change them (doesn't IBM own the masks now?). Also where did you hear about the 486 chip. I hope you take these questions in the spirit they were meant (I don't mean to question your knowledge of these things). It's just that I personally am in a quandry concerning committing my operations to the current slew of 386 machines when I hear talk of 'mask changes', incomplete virtualization, and 486 chips. Jeez Beaver, I just now

get my boss all worked up about a true multitasking OS for the 386 chip and now I hear this! Anyway, I hope you can clear these points up for me.
Appreciate your time,
NOT(fheilbronner)

os386/vm #11, from billn, Tue Nov 18 15:06:20 1986. A comment to message 10.

I'll try and respond to your points. But first a word from... You might want to read grafic.disp/comdex #31 which is a view of the 386 trends I just posted. A few messages earlier is my overview of the Comdex show. (Brought to you by BillN, moderator of grafic.disp). Now back to our program. 1. 386 virtualization not complete. The 386 cannot virtualize itself (run 386 protected code under a VM operating system) nor can it fully virtualize protected 286 mode due to a few critical OS type instructions (forgot the exact ones) which affect the OS but cannot be forced to cause an interrupt if run under a vm type OS.

2. 386 masks being changed. In order to fully virtualize the 286 and 386 protected modes, the 386 chip masks would have to be changed, causing possible six to nine month delays while the new chips are made, tested, proven and delivered. This delay would be unacceptable to most people. IBM has the rights to copy the chip or modify it for their own use. This does not affect Intel's right to make further changes.

3. 486 chip. Intel has admitted that a 486 chip is under development, though perhaps not officially. It is a logical progression and fully expected by the micro community.

4. True multitasking OS. Yes, with limitations. I'll be trying to put together an overview in the next week or so on the 386 OS arena. Stay tuned. (Don't Panic - A. Dent)

BillN

os386/vm #12, from fheilbronner, Tue Nov 18 17:26:48 1986. A comment to message 11.

Thanks for the comeback Bill. Panic is my middle name. I guess I'll just hold on to my obsolete <grimmace> 286's. It's not that I'm trying to be just another computer jock with a nitro-methane box, but I've got a bunch of systems in their senior year at DOS-U. and I can't decide whether they should try and get jobs in the *REAL* world or go for their Masters at the U of 386. I'd give up my dads slide rule if my boss would take off his blinders and see that we (the DOSers) have just gotten too big for our britches. I need multitasking but I've got to sneak it in as a PC due to the fact that the mere mention of *NIX around here is the kiss of death. Anyway, thanks agin for your time, and I'll stay tuned for further developments...

NOT(fheilbronner)

RUMORS ON INTEL 8Ø486

os386/vm #14, from dtuttle, Tue Nov 18 22:33:29 1986.

Information which I have seen/heard on the '486 indicates that it may start to show up in products in the first part of 1990. The targetted differences will be upgraded semiconductor technology, more pipelining / parallelism, prabably separate data & instruction bus, and at least some cache on the chip itself. Initial parts would run in the 24 MHz range, with technology targetted to peak at 30-32Mhz. Clearly all of this info is subject to change and was hearsay to begin with...

os386/vm #15, from dtuttle, Tue Nov 18 22:36:12

I forget to mention that Intel is committed to a '486 which is _fully_ software compatible with the 80386. They expect to be able to run the '486, with a small adapter board, plugged into the socket for a '386.

386 SO FAR ...

os386/vm #19, from tpereira, Wed Nov 19 19:48:42 1986.

These are my first impressions on the 386:

The overall feel and look of the machine is of being sturdy and well built. The keyboard resembles a VT220 format, and it has very good response. The previous machine I was using had a tendency of getting the return key stuck. (Yes, this is already my second 386, I love to brag about this ...). Reset switch is still missing, but my guess is that the thought was deliberate, because if you are using the machine for development as I do, then you're also clever enough to install one, or buy one, thus not allowing for someone else to come by and do a floppy boot, in case you have one. This situation can be very serious in business environments. My recommendation would be to include one reset switch anyway, but lock it. The convenience just offsets the disadvantages. Now, about that lock: You can't get to the machine without losing some eight screws on the back. Then you slide the top panel up all the way to the front. Same way to insert it back, otherwise the little guide at the bottom will get on the way. And screw it back to place again. No change from the old PC's. I can get to the guts of my VAX 8300 a LOT easier then this, same goes to MicroVax II. Again, I would recommend a flip cover top, and a lock, if you are thinking of secure business environments. I have a lot to say about this, but being here neither the place or time. It seems to me the obvious choice, in a time of loaded up super utility boards, extra add on cards, memory, etc. to have easy and quick access to the inside of the machine. Operations are fast and it feels and really acts as one being in front of a mini-mainframe. Loading and saving files is quick, and the screen scrolls fast. I have installed the Compaq EGA and Compaq Color monitor. I tried to see who made the fast hard drives, but all I could find was the manufacturer of the 360K drive, which is Mitsubishi, same maker of the color monitor. The drives are on top of good sized shock mounts. Formatting a tape cartridge takes forty minutes, so I decided to do my next and first hard disk to tape backup over the weekend. In the meantime, I'll take off ... I tried to boot PC DOS 2.1, just for the compatibility of it. I don't know if it is supposed to run it or not. I assume yes. It does not see the hard drives, the error message being 'Invalid Drive Specification'. DRI Concurrent DOS v4.1 won't boot either: 'Incompatible Rom Version'. I happen to like concurrent.

Turbo Pascal V3.01A runs and compiles without problems. It takes me around ten seconds to compile a 3000 line source program. That feels good. Sometimes the system hangs if the option compile and run is chosen, but recompiling again always seems to restore things back to normal. I dont't know the cause for that one either.

Finally, VTERM terminal emulation showed no problems so far. I am going to try LatticeC next. Things are hectic here, and that's only when I hear Stevie Nicks that I seem to realize that I am living in California ...

Regards.
Tpereira.

os386/vm #20, from dondumitru, Thu Nov 20 02:30:04 1986. A comment to message 19.

>...does not see the hard drives...

You formatted the hd(s) with 3.x, right? Then 2.1 doesn't recognize them because the medie descriptor byte is wierd (it specifies 2k clusters, and DOS 2.1 expects drives that big to have 8k clusters only).
Donald

os386/vm #53, from barryn, Fri Dec 12 01:02:19 1986. A comment to message 19.

> about 10 seconds to run a 3,000 line program through Turbo....

continued

Interesting. It takes about 10 seconds to compile a 3,000 line Turbo Pascal program on my 12Mhz AT clone. And the PibTerm program, which is 37,000 lines (about 1 1/2 Megs) of code, takes 3 1/2 minutes to compile at 12Mhz. I was hoping for more of a speed increase in the 80386 than you describe.

---

os386/vm #60, from tanj, Sun Dec 14 04:50:45 1986. A comment to message 19.

Possible reasons for discrepancies in line rate: - density of comments - number of include files - style of type usage, intensive vs. occasional - overall bytes in files, and speed of drives they are read from/written to. - version of Turbo (inc. BCD/87/soft float) - options in effect (stack check, .com or RAM output). - whether main file is in edit buffer before starting. It is an interesting exercise. Raw specs suggest the 16 Mhz 386 should run about 40% faster than the 12 Mhz 286, assuming neither have wait states (note that Turbo uses all 16 bit code: you would get more like 2:1 on a 32-bit arithmetic-intesive program, and significantly better again if you could compare the old "large" model with the new "32 bit instructions, small" mode). To test just the CPU's it would be ideal to run the same compile out of RAM disk with all swtiches and procedure agreed. I remember 10 years ago the Cray 1 compiled Fortran at 50,000 lines per minute. That looks just about within reach when we get compilers running in "32 bit small" mode.

VIRTUAL 8Ø86 ENVIRONMENT

os386/vm #22, from mjk, Wed Nov 26 17:58:36 1986.

By far the best way to make use of the power of the 386 seems to be to use virtual 8086s almost exclusively, and thus never have to write '386 programs at all (except for the OS). Several manufacturers are, of course, doing exactly this sort of VM thing, but here are a few points I would hope to see covered:

1. Access to paging. Take a DBMS for example. At present, these do their own buffering, using their own algorithms to decide what buffers to flush and which ones actually need to be written to disk. The first obvious reaction is to get rid of all this on the '386 and use the paging mechanisms instead. The trouble is that the underlying paging algorithms won't necessarily know what the records being paged in and out mean, nor which ones will be needed again soon: things that the DBMS has a better chance of knowing. Suggested solution: allow access to the paging mechanisms by means of extended DOS function calls - so that the fundamental paging operations of "Mark page as in use", "Mark page as altered", "Flush out a page from real memory" can be intercepted by the DBMS and special knowledge used where appropriate. For instance, the supervisor can signal an interrupt in the virtual 8086, with a default action of IRET to use the default paging mechanisms. All this is something that Microsoft probably can't do, because they already have Protected Mode DOS, and so have to try and encourage people into that direction.

2. Access to process creation, spawning, and interprocess communication. Again, the point is that this can be done on virtual 8086s without having to rewrite the entire program to conform to stringent segmentation, gate, privilege level conventions. Note also that one can now implement the Most Elegant Fork Ever. When a fork occurs, you mark the whole of the virtual machine's memory as read- only; each time the *first* write occurs to a particular page, there'll be an exception (in the 386, not the VM), and that can *then* be handled by making two copies of the page in question, one for each of the twin VMs, and marking them as read-write. Many parts of the machine, such as MSDOS, will never be written to and hence will never need to be split, saving physical (or paged) memory.

If you make these things accessible to virtual 8086 processes, then it will be quick and cheap to take advantage of the new features of the 386. We can go on writing in Turbo or C or Assembler with only a few tiny subroutines specially written for the 386 environment. Stick to the compiler bugs we know and love instead of discovering new ones.

But the VM suppliers do need to get together and agree some conventions!

---

os386/vm #24, from tanj, Thu Nov 27 20:21:25 1986. A comment to message 22.

The best way to use the power of the 386 will be to write 386 code. Virtual 8086 will just be a transition phase, like when everyone first used the IBM PC in small mode with object-converted Z80 code. Considering that IBM seems uninterested in the 8086, and that any socketed 286 chip can be swapped for a 386 on a small carrier board (it is pin-strappable for socket compatibility) within the next few years as the price falls, it will eventually be a 386 native world as far as first-class programs are concerned. People will just use the virtual 8086 mode to run programs they bought in past years. Note that the "Most Elegant Fork Ever" is usually called "copy on write" and is the norm on BSD 4.2/4.3 or Unix 5, as well as several other operating systems.

---

os386/vm #28, from spiv, Sat Nov 29 02:26:40 1986. A comment to message 22.

Your comments about virtual memory management are correct. Actually, Xenix/386 already does what you suggest. It is called "copy on write" and allows all Xenix processes to share memory pages until a process writes to a page. Then the page is made private to that process only with a virgin copy kept to be shared by the other processes.

---

os386/vm #26, from mjk, Fri Nov 28 08:28:01 1986. A comment to message 24.

>Everyone first used the IBM PC in small mode with object-converted Z80 code.

No such thing: it was possible to *source*-convert 8080 (assembler) code; not object-convert (this would imply reliable disassembly) and never Z80. So conversion involved *work*. The only source-converted product that is still being sold (as far as I know) is Wordstar: even version 1.4 is full of LAHF instructions - but I think it's rather sad. Every context-switch takes ages because no-one knows that the XCHG SP,[mem] instruction exists.

So in 8080 -> 8086 there was a choice: do work source-converting your programs (lots of pitfalls even with DR's XLT86, and impossible if Z80), or do a little more work and write a nice new program. In 8086 -> 386 there is a choice: rewrite the whole of your program to run in native 386, or rewrite just a few modules to run in virtual 8086. The gap between the approaches is *far* wider.

---

os386/vm #29, from tanj, Sun Nov 30 06:01:49 1986. A comment to message 26.

The standard MS Basic (as shipped with DOS) was also a converted program, though it did have some native 8086 sections patched onto the CPM design. It might still be. I accept your analysis that it was source conversion and not from Z80: I moved to the 8086 from minicomputers and never used the 8-bit CPUs, just saw the converted code as it was evidenced by disassembly in 1981/82. But I think the mechanics of the process are not the point, the results are. Lotus 1-2-3 was one of the first programs to make use of the new architecture, and it obsoleted 8-bit spreadsheets overnight. It is true that you can move your existing applications to virtual 8086 with minimal trouble. However, your market may not port with you if someone develops a next-generation product in your market which can use 32-bit data spaces, unconstrained paged code, clean multi-tasking, and port to nearly any 32-bit mini or micro on the market (remember, there are hundreds of thousands of those already out there, some selling like hot-cakes). It may not happen, but then again it may.

WHAT CPU IS THIS?

os386/vm #23, from mjk, Wed Nov 26 17:59:26 1986.

Ways exist of deciding whether a CPU is an 8086, 8088, V20, V30, 80186, or 80286 - which, most of the time, is pretty uninteresting information anyway. But on the 386, we have the 66H (Operand Size) and 67H (Address Size) prefixes, two extra segment registers, extra logical instructions, and 32-bit arithmetic, and these are such a good thing that it'll often be worth providing both 386 and 8086 versions of critical routines (eg arithmetic) and deciding at run-time which to use. So we need to decide at run-time whether this is a 386.

On the 386, the use of the LOCK prefix is severely restricted, and most useless combinations (such as LOCK / INC AX) will cause an exception 6 to be signalled. In Real Mode, then, you trap interrupt 6, issue a useless LOCK, and see whether INT 6 was signalled. If not, 8086; otherwise 386.

In Virtual 8086 mode, there is a problem. The exception will still be signalled, but the VM supervisor will see it. If it is an exceptionally kind and friendly VM supervisor, it may decide that the instruction that caused the exception should be allowed to execute as if nothing had happened. If it does this, then there'll be no way to tell a virtual 8086 from a real one, and so no way to decide whether all these nice new instructions and addressing modes can be used.

So please, VM manufacturers, don't be too user-friendly. If you were thinking of hiding "illegal LOCK prefix" exceptions from us, leave at least one of them visible, and tell us which one it is!

os386/vm #25, from tanj, Thu Nov 27 20:21:28 1986. A comment to message 23.

You can try 32 bit arithmetic instructions, using prefixes, to tell if you are on an 80386. Even in virtual 8086 mode these are available. However, on an 80286 even in real mode they will trap as illegal ops, so you must plan on catching the exception in case it is an 80286.

os386/vm #27, from mjk, Fri Nov 28 08:28:30 1986. A comment to message 25.

>Even in virtual 8086 mode 32-bit instructions are available. Yes, I know: that's why we need to distinguish a virtual 8086 (or real-mode 386) from the real 8086 and 80286.

os386/vm #32, from tpennello, Tue Dec 2 04:22:50 1986. A comment to message 23.

From my discussions with people at Intel and Compaq, there is no way to sense 386 vs. xx86 without getting a protection exception in protected mode. However, if you're running under MS-DOS, this doesn't matter. Here is a routine from compaq that detects 86/286/386 in real mode: pushf; xor ax,ax; push ax push ax -- try to put 0 in flags popf; pushf; pop ax and ax,0f000h; cmp ax,0f000h; je is_86 mov ax,0f000h; push ax; popf; pushf; pop ax and ax, 0f000h; je is_286 -- 286 clears upper 4 flag bits -- if we get here, it's a 386 since the upper 4 flag bits remained on is_286: -- it's a 286 is_86: -- it's a lowly 8086 I've used such a test for discovering presence of a 386 in a recent modification to the High C run-time library. When the 386 is there, I'll use "real" instructions to do 32-bit division rather than more expensive software emulation. Tom Pennello, MetaWare

os386/vm #33, from intel, Thu Dec 4 01:43:36 1986. A comment to message 32.

Thanks Tom for posting the how to figure out what processor you are using code. I was digging around looking for my memo I sent to Compaq many months ago.

I would like to point out that the easiest way figure out what processor you are using is to have the OS store the information about the processor somewh ere in memory and/or make a processor Id system call available to application programmers. This is real nice on the 386 because after reset the 386 stores the component and revision (i.e stepping id) in register DX. I have ask MicroSoft and Compaq to make this available to application programs. I'm not sure if they are going to do this or what effect this has on application programs running on non-386 systems. But I tried.

Clif Purkiser Intel

os386/vm #34, from dondumitru, Thu Dec 4 03:10:40 1986. A comment to message 33.

I agree - we are getting to the point with 80x86 where a standard system call to determine cpu type would be very handy. And it should be one of the easiest things to provide (on the same level as determining OS version).
Donald

os386/vm #35, from tpennello, Thu Dec 4 06:17:48 1986. A comment to message 33.

I would prefer an instruction to determine it, so one doesn't have to depend upon a particular operating system. There are plenty of 0f opcodes left -- why doesn't Intel start now installing a "get cpu id" instruction? With the instruction, we only have to depend upon Intel to do it right. With the OS call, we have to depend upon every OS to do it right.

os386/vm #36, from intel, Fri Dec 5 03:02:54 1986. A comment to message 35.

I menitioned a similiar idea to the architects about year-ago, they left grumbling about lack of microcode space and extra work needed in the instruction decode unit.

I will pass the request along to the 486 boys. I don't about you software types we give you a processor with segments greater than 64K and you want more :-) (the usnet sign for a joke)

os386/vm #37, from skluger, Fri Dec 5 11:05:55 1986. A comment to message 36.

But Real Programmers DON'T want SEGMENTS AT ALL...

os386/vm #38, from billn, Fri Dec 5 23:19:07 1986. A comment to message 36.

While you are passing notes to the 486 guys, let me suggest one of the real concerns with segments. Performance. As a potential solution, how about a segment register cache? Maybe extend up to 8 segments with perhaps 32 or 64 cache entries. Since cache is a regular design, it would not take as much real estate.
BillN

os386/vm #39, from tanj, Sun Dec 7 06:01:16 1986. A comment to message 36.

If the architects threw out the segments entirely for 32-bit mode then surely they would have the room to add a few instructions and some cache... Does Intel really expect the 32 bit segmentation to be important ? Do any of the operating systems currently in production run applications except as separate paged linear 32-bit address spaces ? Will the 486 be like the 286 ? Lots of bells and whistles designed in before it became evident the market was actually going to use the 386 in the fastest and most obvious way, unrelated to the fancy extensions ?

os386/vm #40, from jshiell, Sun Dec 7 19:04:28 1986. A comment to message 38.

In native 386 mode how many segments do you think are going to be used. If the model you "Like" is anywhere near flat (FAST) then a segment register cache would not buy you very much. A better buy would be a "Loop

continued

mode" in the prefetcher where it "Knows" about last (1 to maybe 4) LOOP or JCXZ ops. and starts prefetching down their taken paths. It is a very simple form of branch prediction. Jon Shiell

LOCK PREFIX

os386/vm #31, from jshiell, Mon Dec 1 21:38:53 1986.

Does anyone know where/when DOS, Xenix etc. use the loxck prefix ?? Jon Shiell

os386/vm #46, from villi, Tue Dec 9·12:37:19 1986. A comment to message 31.

My bet is that none of them ever uses the LOCK prefix. The LOCK prefix is used to secure the system bus agains access from co-processor and other directly connected devices. It is not needed with the 8087, and the only other PC hardware I can think of that might cause contention for the bus is DMA, but I think that's managed by the DMA controller. So: most likely, the answer to the question is NEVER, NOWHERE.

os386/vm #47, from skluger, Tue Dec 9 14:11:30 1986. A comment to message 46.

Looking at the IBM AT schematic, I envision electrons dripping from the LOCK pin on the 286... It's not hooked up!

os386/vm #48, from mwaite, Tue Dec 9 21:25:33 1986. A comment to message 47.

So you can't easily run multiple 286DOS programs on the 386 in VM. Okay but is that a problem. For example, what KIND of program would you expect to run specificaly for the 286DOS rather than DOS3? Well one that uses the >640K barrier. Which will be lots of stuff from super fast RAM based word processors, database engines, etc. So I own two of these programs. I buy my 386 box which claims multiprocesing ability, boot up the VM386DOS, run my two programs and I get: Sorry only one 286DOS program allowed at one time. Is that it?

os386/vm #49, from geary, Wed Dec 10 01:50:12 1986. A comment to message 48.

Not really. If you get 286DOS running on the 386, it will run multiple 286DOS programs just fine.

For my money, I'm waiting for 386 Windows...

386 CAN'T VIRTUALIZE 286 APPS?

os386/vm #41, from mwaite, Sun Dec 7 21:38:16 1986.

This may have been asked before, but a recent InfoWorld article claimed that the 386 can't virtualize 286 programs, meaning code optimized to the new 286 won't work without modifications on the 386. Is that true?

os386/vm #43, from mjk, Mon Dec 8 03:25:51 1986. A comment to message 41.

>Code for 286 won't run on 386... As I understand it, it will, but only on the real 386 - ie on the whole machine; so while you can run a dozen virtual 8086s on one 386, you can only run one 286 & it won't co-exist with anything else.

os386/vm #44, from geary, Mon Dec 8 04:57:04 1986. A comment to message 43.

That's right. For example, programs written for 286DOS will work fine on the 386. And 286DOS itself ought to run OK, with maybe some changes in the initialization code. But you couldn't run multiple 286DOS's under any kind of VM 386 system the way you can run multiple DOS 3.2's.

os386/vm #45, from bilin, Mon Dec 8 11:03:23 1986. A comment to message 44.

Actually, while what you said is accurate, there is a way to run multiple 286 code systems on the 386. The problem resides in one or two instructions in the 386 that cannot be virtualized. There was some discussion at the vm demo about changing the 286 code to avoid use of those instructions, making it possible to virtualize the 286 code successfully. BilIN

I/O PROCESSOR?

os386/vm #50, from peddy, Thu Dec 11 13:02:34 1986.

In the recent rash of announcements of 386 add on boards for AT type machines, I expected to see at least a few that would use the present 286 chip as an i/o processor, especially on Intel's 386 board! Doesn't this seem like an obvious thing to do?

As for the benefits of using i/o processors on microcomputers, witness the old Compupro 10: The system contained an 8088 and four Z80's. Under Concurrent DOS, The 8088 ran all the 16 bit stuff while the Z80's functioned as i/o processors when they weren't running 8 bit software. The result was that dBase II ran faster on the Compupro than on an single user AT!

With the new control programs for the 386 and the memory management features of the 80n86's, I can only assume it would be almost trivial (or at least not too hard) to do the same thing with an AT... or am I living in a dream world?

os386/vm #54, from villi, Fri Dec 12 17:31:13 1986. A comment to message 50.

Another view of i/o co-processors: The DEC Rainbow has both a Z80 and an 8088. When running MS-DOS, DEC literature proudly points out, the Z80 is used as an i/o coprocessor, freeing (supposedly) the 8088 up to do other tasks. Well, if that's true, I can't imagine how slow the Rainbow would be WITHOUT the Z80, since it is ABSYMALLY slow at disk and diskette i/o; certainly much slower than the IBM PC, which does not sport an i/o coprocessor. So, folks, keep in mind that what the salesman says isn't always the whole truth.

os386/vm #59, from rnelson, Sat Dec 13 14:04:55 1986. A comment to message 54.

The reason the DEC Rainbow as so slow at I/O was NO DMA. The processor had to read from the disk byte by byte. Yuck.

os386/vm #67, from ronlepine, Mon Dec 15 11:04:12 1986. A comment to message 59.

No DMA in itself does not have to be slow. The TI Professional has no DMA chip for disk i/o but does as well as IBM on many tasks. Some are faster and some are slower. That is if you use disks formated by a TI. Using disks formated by a IBM is the surest way to convince someone the TI is SLOW on disk i/o.

os386/vm #70, from jgotwals, Thu Dec 18 19:05:32 1986. A comment to message 59.

The IBM AT doesn't use DMA for HARD disk io. It reads from the disk controller sector buffer word by word!

os386/vm #74, from tanj, Sun Dec 21 04:49:34 1986. A comment to message 70.

In PCDOS using rep insw is a rational decision, since it moves faster than an 8237 and there is nothing else to do while waiting. As for multi-tasking systems, the only rational way to do it is with a better controller board that just takes the command, delivers the transfer, and interrupts (or chains to the next command) when everything is wrapped up. It amazes me that IBM originally hyped the AT as a multi- user machine: after all their mainframes may have lousy instruction sets but they did a first class job on IO architecture, so they do know how it should be done.

---

os386/vm #75, from ksarno, Sun Dec 21 12:33:51 1986. A comment to message 74.

I would like some clarification on this point, Mr. Tanj, if you wouldn't mind. It seems to me that the disk controller DOES fetch the sector off the disk (into its local buffer) and then interrupt. The operating system has to move the data from the controller to its own buffer (and possibly again thereafter to a user buffer), and that is unfortunate but hardly unusual in the micro world. But given that fact, there is nothing worse about doing a 16-bit-wide rep insw from the controller buffer than 8237 DMA. The data has to be moved, and the insw is the fastest way to move it and the cheapest.

However, I think the insw is the most desirable way to move disk data under ANY operating system for another reason. With DMA, the disk move becomes the highest-priority processing thread in the system. It WILL steal cycles (either all of them or half of them) regardless of what else is going on. If, say, a character interrupt from a UART comes in and real-time response is important to avoid a 9600-baud data loss, the response to the character interrupt WILL be slowed down by any disk DMA that is going on at the same time. However, the string input is fully interruptible. If a character interrupt must be processed, the the string move stops until the end of the interrupt processing, causing no delay to the real-time interrupt code. It is right that disk IO processing should be subordinated to certain other, higher-priority events. This can't always be done with DMA, but it CAN always be done with a string move.

---

os386/vm #76, from billn, Sun Dec 21 13:17:22 1986. A comment to message 75.

I think you overlook the point about multitasking. If the cpu is busy moving data, no matter how fast, it cannot be doing real work for one of the tasks, and therefore is slower than a DMA alternative.

There is also a problem raised by queuing theory that indicates a bottleneck caused by requiring two servers in sequence before finishing a task.

These problems mostly don't show up under single task or non disk intensive workload, but multiple users on UNIX will clearly see the effects, though masked by the built in cache.

And, BTW, the crippling of the AT by IBM was probably not accidental. They had their base of S34 and S36 to protect at the lower end. BillN

---

os386/vm #77, from tanj, Tue Dec 23 18:02:58 1986. A comment to message 75.

I think you will find that DMA is subordinate to the CPU, since the 8237A must get permission from the CPU DREQ pin to grab the bus. Comms are handled by the CPU and therefore not threatened. The reason it is done like that is exactly to prevent the situation you describe and ensure that maximum latency can be guaranteed in such systems. Actually the Intel CPUs are pretty greedy: in one board I worked on we solved a latency problem in a DMA channel by making sure the CPU did a true HALT rather than an idle loop in the background. The loop caused continuous instruction fetch, worse even than a typical computation sequence (the 8086/8 CPU fetches waste bytes beyond the end of a loop even after a jmp, the pre-fetch engine seems rather like the hind-brain of a dinosaur which takes a while to change direction) and a low priority DMA never made it through. It can happen that comms fed in on another DMA channel could be hurt, and that would be relevant if the PC had smart comms chips like the Zilog ASCC's, but it doesn't. The ideal is to use channel structures such as Intel is slowly moving across its product line (starting with the 82586 Ethernet controller, then a text chip, recently the 82786 Graphics controller) where the CPU can just chain commands and the chips have built in DMA channels. The multi-tasking CPU just does a clean task swap and gets on with useful work with no time taken in waiting. Eventually an interrupt will signal completion and the interrupt routine moves the related task back on the ready queue. It is up to the bus arbitration circuit to ensure no DMA or CPU can hog the bus: a sophisticated design problem, but one familiar to any good hardware designer. BTW, its Mr. Bennett, Tanj is my first name.

---

os386/vm #78, from ksarno, Thu Dec 25 01:51:57 1986. A comment to message 77.

Oops! Excuse me, Tanj, for the error in your name. But I still maintain that DMA can and does degrade the response to communications interrupts on the IBM PC family. It is true that the 8237 must get a Hold Ack from the CPU (DREQ comes from a device into the 8237) before it will take over the bu s. But according to the 8086 data sheet, the 8086 will give up the bus at the end of the current cycle or the next cycle in most cases (exceptions are locked instructions and interrupt acknowledge sequences) when HOLD is asserted. That is, the 8086 does not refuse to get off the bus for very long if at all. Once the 8237 has the bus, of course, it can keep it for as long as it likes just by keeping HOLD asserted. For example, a nasty disk driver writer who chose to use DMA could program the 8237 to do Block Mode DMA transfers, in which it would hold the bus for a full 512 bus cycles without ever giving it up, once it got control. The CPU wouldn't get ANY instruction fetches in this case. Fortunately, most people aren't that crazy; they program the 8237 in single-transfer mode, in which it drops HOLD after each bus cycle, allowing the CPU to get back in for a cycle or two. But the 8237 still WILL get in for one cycle out of every two or three, meaning a bus cycle reduction of 33% to 50% for the CPU. I think the loop problem you cited may be an undocumented special case: the prefetcher refusing to give up after a jump instruction empties the queue. But most code doesn't consist of tight loops, so there will be significant degradation unless most of your code consists of multiply and divide instructions and loops.

In any event, Merry Christmas.

---

os386/vm #79, from tanj, Sun Dec 28 05:12:57 1986. A comment to message 78.

Hmmm, our hardies simply told us "thou shalt not block mode" and we obeyed, since the hardies had designed all their devices with correct buffering to operate in single-transfer mode. Unfortunately the PC is a jungle where one encounters dubious devices and software, though the benefit has been the extreme fitness of the best programs evolved in the jungle. The 386 is a protected machine, and (unlike the 286) will run protected even while DOS is supported, so I was going to write that the problem would go away. But of course device driver software will be loadable (to do otherwise would be a suicidal freeze-out of add-on devices) so there will still be no control of what kinds of hard and soft tricks are perpetrated. Use of a block mode DMA would probably be fatal because any such product would be drowned in a flood of complaints and bad reputation from its unlucky first users. It will be normal however to take the maximum permitted in single transfer mode for transfer to and from device buffers. The only redeeming feature is that the end user of a PC normally wouldn't be doing anything, including most comms, which a 386 couldn't handle while 50% of the bus cycles go elsewhere for a few milliseconds every now and then. Maybe you are right that there was a bug ("undocumented special case") in the 8086 pre-fetch. At the time we were busy and mighty glad to solve it. Seasons greetings ! bye

---

os386/vm #85, from qnx, Mon Feb 2 21:58:25 1987. A comment to message 75.

I know this is an old thread (DMA vs. Processor controlled movement of disk data), but I had to throw my 25 cents (inflation has hit) in. QNX (the O/S) does not use DMA for the reasons mentioned in #75 (real-time

continued

performance, etc). In fact, file I/O is at priority 3 with device I/O at one level higher (priority 2) and time slicing and memory management (among other things) at priority 1. It works very well as evidenced by the number of customers doing real-time operations such as data aququisition.

VM/386 WHAT IS IT?

os386/vm #68, from softguard, Wed Dec 17 20:01:00 1986.

VM/386 is a control program designed exclusively for 80386-based computers. It's patterned after the VM/370 operating system for IBM mainframes, but is oriented to the microcomputer environment. The main goal of VM/386 is to deliver multitasking, virtual memory, multiuser capability and a growth path to future hardware/software without forcing the user to change or repurchase any software. VM/386 creates a set of virtual machines in the Virtual 86 Mode of the 80386. Each virtual machine acts like a real computer and thinks it has exclusive access to all of the resources of the real computer. Virtual machines are protected from one another; a 'crash' in one virtual machine does not 'crash' the entire system. Each virtual machine can have its own operating system, application, and terminate-stay-resident programs.

```
===========     ===========          ===========
|         |     |         |          |         |
|         |     |         |          |         |
|Lotus 123|     |dBASE III|          |         |
|         |     |         |          |         |
|         |     |         |          |         |
|---------|     |---------|          |         |
| SideKick|     | ProKey  |          |---------|
|---------|     |---------|          |Proprietary|
| DOS 2.0 |     | DOS 3.1 |          |   O/S   |
===========     ===========          ===========
     |               |                    |
=================================================
|             VM/386 Control Program            |
=================================================
                       |||
=================================================
|80386 Hardware (CPU, monitors, disk drives,etc)|
=================================================
```

Multitasking is obtained by runnning different applications in each of the virtual machines. Different virtual machines can also run the same application but with different data. Any operating system or application that runs on an IBM PC or PC-compatible can be run from a virtual machine under VM/386. You can 'hot-key' from one application to another. The background tasks continue to run while the foreground task talks to the real monitor and accepts input from the keyboard. Since the background tasks write to virtual devices, VM/386 can multitask applications, such as Lotus 123, that are not 'well-behaved' and write directly to video RAM.

The prime assumption behind VM/386 is that operating systems (such as MS/DOS) and their applications have a delicate, tightly bound relationship. Any changes in this relationship can be hazardous. Most successful PC programs aren't "clean", they directly use hardware and/or take advantage of "undocumented" DOS features. This leads to a great many difficulties when attempting to move such applications into newer OS's. It should be pointed out that the reason many applications are "dirty" is to deliver performance and features to the user that are otherwise not possible. VM/386 preserves this relationship between applications and OS's by treating them as a matched set. The operating system expects to find certain hardware, memory, privileged instructions, etc. By delivering a "virtual" set of these hardware resources, the entire OS can be "hosted". This hosting isn't limited to a single OS. Multiple OS's (or "guest" OS's) can be hosted simultaneously. VM/386 delivers a "virtual reality" to the OS and its applications (virtual memory, virtual interrupt controller, virtual hard disk, etc.) and maps it to reality. The "virtual reality" and reality can be quite different. In a VM environment, OS's are almost like applications. They can be selected based on comapatability or feature needs on a user by user basis, without forcing the entire machine to run only that particular OS (e.g. Topview, Windows and Gem running simultaneously). MS/DOS is designed as a single user, interactive, fairly simple OS. It's not unlike CMS under VM/370. By running multiple copies of MS/DOS, multitasking and multiuser environments are possible without painful conversions.

Brief Technical Overview.

VM/386 is written entirely in assembler. It runs in native 80386 32-bit mode. Assembler was chosen because of the critical timing problems involved with virtualizing hardware. VM/386 is designed exclusively for the 80386 (and the coming 80486). The heart of VM/386 is the control program (CP). The CP is the kernel or nucleus of the system. CP doesn't involve itself with hardware or BIOS's. Its job is to handle 80386 exceptions (such as page faults and privilege exceptions) and route those exceptions to the proper software handlers. The Hardware Device Driver (HDD) is the primary method of dealing with the outside world in VM/386. HDD's are not unlike DOS or UNIX device drivers in concept. However, HDD's exist OUTSIDE the guest OS. They are responsible for managing and "virtualizing" various hardware components. Such events as "guest OS attempted IN", "guest OS attempted OUT", "real external interrupt" and "virtual external interrupt" are dealt with by the HDD's. Compared to the 370, the PC I/O architecture borders on anarchy. A "device" can be a collection of ports, maybe an IRQ line, maybe a DMA channel, maybe some dedicated RAM or any combination thereof. The HDD attempts to bring order to this chaos. Hard disks may be set up as "minidisks". A minidisk is like a partition, except that the disk cylinder numbers are virtual. That is, each minidisk appears to the guest OS as starting at cylinder number 0. This allows for the simultaneous execution of guest OS's with incompatible file systems. The File Share Subsystem (FSS) manages simultaneous MS/DOS file accesses in the VM/386 environment. MS/DOS has no facilities to allow file sharing and locking across multiple OS executions. The FSS allows multiple DOS's to share a single hard disk in an orderly fashion, preserving file and data integrity. The FSS is a layer of code that is aware of the characteristics of one particular guest OS (MS/DOS) and is also aware that it's running under VM/386, this type of code is referred to as a subsystem.

There will be quite a bit more techincal information posted later on. This is just a brief introduction.

VM/RUN "High" Memory Model.

VM/386 uses MS/DOS as a "starter motor" operating system to get going. This approach was taken mainly to avoid forcing hard disks to be reformatted, and to take advantage of existing DOS I/O capability. During VM initialization, DOS is used for I/O in 8086 mode, then the machine is placed in 32-bit protected mode, to access extended memory, then the machine is returned to 8086 mode for more I/O, and so on. Another way of describing this process is to say that 80386 32-bit programs running in extended memory are using MS/DOS for their I/O. If this technique works for VM/386 initialization, why not for application programs? This observation led to the development of VM/RUN. VM/386, which is a very large system, won't be ready for sale until the 2nd quarter of 1987. However, the part of VM/386 that allows for the use of 32-bit mode and extended memory will be ready at a much earlier date. A Pascal and C compiler for the 80386 (that runs under DOS) is available from MetaWare, an assembler and linkage editor for the 80386 is available from Phar Lap Software. Softguard is providing VM/RUN (for 32 bit environment loading) and an accompanying debugger. As a set, these tools are a complete development environment that allow for 32-bit programming under "stock" MS/DOS. The following memory layout is used for programs that take advantage of the "32 bit mode" of VM/386 and/or VM/RUN.

```
 1023
:-------------------------------------------------------
-:
 MEG  :    386 application program stack. May be at
:
      :       3FFFFFFCh or "top" of application
:
        :       virtual memory or fixed address.
:
        :       Many options.
:
        :
#10:

:-------------------------------------------------------
-:
        :
:
        :       386 application program area.
:
        :       Always starts at virtual address
2000000h        :
        :       (32 meg decimal).  Runs at CPL 3.
:
        :       Default IOPL = 0, may request 3.
:
        :
:
        :
#9 :
32 MEG
:-------------------------------------------------------
-:
                ALL ADDRESSES ABOVE 1 MEG ARE VIRTUAL

 1 MEG
:-------------------------------------------------------
-:
        :       Video RAM, BIOS, etc.
#8 :
        :
:
 640K
:-------------------------------------------------------
-:
        :       DOS free area.  Mainly for use by calls
:
        :       such as 4B.  Optional
#7 :
        :
:
 512K
:-------------------------------------------------------
-:
        :       Area given to high virtual addressing
:
        :       Optional.
#6 :
        :
:
 384K
:-------------------------------------------------------
-:
        :       User managed low buffer pool.
:
        :       Optional.
#5 :
        :
:
 320K
:-------------------------------------------------------
-:
        :       VM/RUN managed buffer pool.
:
        :       Includes default INT 21 DTA.
#4 :
        :
:
 256K
:-------------------------------------------------------
-:
        :       VM/RUN monitor
:
        :       (and debugger)
#3 :
        :
:
 90K
:-------------------------------------------------------
-:
        :       TSR's (if any)
#2 :
        :
:
 50K
:-------------------------------------------------------
-:
        :       DOS area
#1 :
        :
:
  0
:-------------------------------------------------------
-:
```

Diagram Description.

The above diagram shows what a program designed for the 386 "sees" when executing in the VM/RUN environment. All memory addresses below one meg, with the exception of area #6 (low real memory mapped to high virtual addresses) run with V=R (virtual addresses = real addresses). Area #1 is the DOS operating system area. Area #2 is any user device drivers, terminate and stay residents, etc. Area #3 is the runtime VM program and 386 debugger. Area #4 is the "low" buffer pool used to handle I/O requests from up "high". Managed by VM. Area #5 is the user specified "low" buffer pool. This V=R area is used by 386 programs for I/O. By using low addresses, buffer copying overhead can be avoided. Area #6 is low real memory given to "high" virtual addresses (more ram for a 386 application). Area #7 is reserved (via shrinkblock) for DOS to service such things as the int 21/4B function. Area #8 is the videoram and BIOS. Area #9 is the 386 application program. It's always loaded at virtual address 2000000h (32 megabytes). Area #10 is the 386 application program stack. Its size and location are specified at execute time (default is virtual top of 386 program, e.g. 2200000h on a machine with 2 megs of real memory given to the 386 application area).

Notes.

The code model can be thought of as "68000 style". A large flat address space that's very accomodating both to C and assembler programs. No segment registers are used. The 386 application program can be thought of as a "1 gigabyte COM file" using the "small" memory model. The starting address of the 386 application is always 2000000h (this avoids any extended or expanded memory conflicts).

The application code should run unmodified under VM/RUN or the full VM/386. The "16 bit" DOS area will be available to the 32-bit program for inspection/modification. Videoram can be directly addressed without segmentation (e.g. EDX = B0000h to get at monochrome videoram). DOS INT 21 I/O functions are supported. 32-bit int 21 calls will be "deflected" into 16-bit mode for service. EDX will be the buffer pointer instead of DS:DX. No FCB calls (DOS 3 style only).

More details will be posted in the near future.

---------- For More Information ------------

Please contact:

Softguard Systems, Inc
2840 San Tomas Expressway
Suite 201
Santa Clara, CA 95051
Tel: (408) 970-9240

continued

MOST-HYPED CHIP

users386/other #2, from dondumitru, Wed Nov 5 02:07:14 1986.

I feel compelled to nominate Intel's 80386 processor for the Most-Hyped Chip of the Decade Award. Just look at this weeks InfoWorld, not to mention the last three issues, last month's DDJ, this month's PC Magzine, and who knows what else. Granted, I want one on my desk, running multiple operating systems, and all that nifty stuff. But this is getting ridiculous, considering that I know of *one* company that is shipping the machines. Donald

users386/other #3, from tpereira, Wed Nov 5 13:29:23 1986. A comment to message 2.

I agree mainly with you. But also is the factor that something is being done. I have a 386 Compaq, it is a splendid machine for a price that I cannot afford. $9045 w/ EGA and colro monitor. Undoubtedely, the 68000 is a much better chip without a much less Hyped trail. Regards. TPereira.

users386/other #4, from jgotwals, Wed Nov 5 19:54:31 1986. A comment to message 3.

The extreme interest in the 386 chip over the 680x0 is not because one chip is better than the other, but it is because of the massive amounts of PC DOS software which the 386 can run.

80386 LIFE CYCLE

users386/other #9, from dwarren, Sun Nov 30 21:00:30 1986.

I am making a private study of developments revolving around the 80386 chip. From press reports, I've put together the following table of significant events likely over the next 3-4 years. The table assumes (1) a 3-4 year life cycle for the 80386, and (2) the implementation of both 386-based hardware and software in a second-generation form (e.g., per rumors about IBM's likely offering) after the first-generation hardware (Super AT) and software (Advanced DOS 1.0 and Unix systems).

Life Cycle: 386-based Products

Time	Development
2H 1986	o First Super AT's announced and delivered o Uncertainty and confusion among hardware supplier surrounding standards o Several first-generation multitasking operating systems announced
1H1987	o First-generation multitasking operating systems delivered o IBM announces second-generation 386-based machine
2H 1987	o Microsoft Advanced DOS 1.0 for protected mode of 286 delivered o Major software firms announce intention not to rewrite software for 286 protected mode o Super AT's enter commodity marketing phase o PC's reach retail stores as low-end consumer product; 286-based machine becomes PC standard, AT phased out as high-end machine o First IBM second-generation 386 machines delivered o First clones of IBM second-generation design announced, for delivery next year
1H 1988	o First second-generation clones delivered o Sales of 386 machines overtake 286 o First 386-specific software available
2H 1988	o Second-generation multitasking/mainframe networking) operating systems delivered
1H 1989	o New 32-bit chips sampled (80486?)
2H 1989	o Large-scale network interconnect systems implemented, based on second-generation operating system o New types of add-in boards for second-generation machine appear
1990	o First machines based on new 32-bit chip introduced

Would greatly welcome horse laughs or comments of any other kind about the timetable proposed here. David Warren

users386/other #10, from skluger, Sun Nov 30 21:16:26 1986. A comment to message 9.

You asked for it you got it...

2H 1987

o MC68030 gives Motorola total dominance of the 32-bit market
o 80386 succumbs as Intel cannot get a significant yield of 25 MHz chips and is unable to supply 80387 in any quantity

Should the 386 be a moderate success, you will be right about the first appearance of '386-specific software in 1/88. I predict that the 386 will gain a strong foothold in the OA market doing minor spreadsheet-type number crunching, data base applications and word processing. The scientific workstation market will be divided between National and Motorola, with the latter gaining on the former.

users386/other #11, from billn, Sun Nov 30 21:20:28 1986. A comment to message 9.

At this point I can't see where there is a big blunder in your schedule. It is possible you may be too conservative. And graphics displays and applications will be a major factor in the market - don't forget their impact. BillN

users386/other #12, from johnf, Mon Dec 1 03:39:44 1986. A comment to message 9.

I would think that 386 specific software will become available in late 1987 rather than 1988, but then I have always been an optimist. What is your prediction for add-ins for first generation 386 machines? Also, I would be interested in the availability of an industrialized 386.

-:- JohnF -:-

users386/other #13, from dwarren, Tue Dec 2 15:54:51 1986. A comment to message 12.

I'll tell you what I think about add-ins for first- generation 80386 machines, and I'd like to hear what you and others think.

I think add-ins for first-generation 386's are here now: the boards designed for the PC and AT buses. Other possibilities are limited by the lack of bus standardization and the use of 16-bit I/O in the first-generation designs.

One type of board that may gain popularity in first- generation machines is a communications and emulation board that supports multiple sessions or that performs functions previously requiring separate boards. Otherwise, each concurrent session could require one board. Arnet markets an intelligent communications board called Smartport that turns a PC into a Xenix multiuser system. Available now in 4- or 8-port models, it features 64 Kbytes of RAM that can be accessed by both the computer's main processor and the 80186 coprocessor on the board. This board takes over I/O processing to support the multiple ports, freeing the main processor for computing and reducing degradation in multiuser performance. It uses the I/O channels of the PC bus. Other firms that supply multiple-session network boards for PC's and that have announced the intention to provide similar products for 386-based PC's are CXI and Digital Communications. Other makers of multiuser add-ons for PC-AT's may offer similar products, including Alloy, HAAR Industries, and Software Link. Looking ahead to a second-generation design, memory boards will continue sell well. The 1 megabit DRAM's, 8 MB will fit in a single slot. However, until there is standardization in the 32-bit memory bus and memory organization (PCET standard?), there can be no add-in 32-bit memory boards, save those made by the manufacturer of each machine. Developments in graphics cards are mainly speculative at this point. One is higher resolution, needed to support computer-aided design and engineering applications. A second is the use of the 32-bit bus to transmit signals faster than possible on 8- and 16-bit buses (this is not for the first- generation machines, though). A third is to use the speed of the 80386 to take over more graphics processing, simplifying and reducing the cost of electronics on the graphics board--in the extreme, entirely eliminating the graphics card. Still another, more exotic type of add-in board will provide more capable interfaces between computer and external world, such as drivers for speech recognition and machine vision devices. I, too, would be interested in the availability of an industrialized 386. Anyone have any insights? David Warren

users386/other #14, from dwarren, Fri Dec 5 20:30:45 1986. A comment to message 11.

Graphic displays and applications a major factor Thanks for comments on 80386 timetable. Can you expand at all on your comment about the role graphics displays will play in the market for 386 products? What role? When? Any thoughts will be welcome. David Warren

LARGEST ALLOWED INSTRUCTIONS

users386/other #17, from jshiell, Fri Dec 19 01:51:01 1986.

From the FYI dept:

The 8088/8086 have no limit on the size of an instruction.
The 80286 limits an instruction to 10 Bytes max.
The 80386 limits an instruction to 15 Bytes max.

Thus two instructions (an 11 byte followed by a 16 byte) could be used to tell which of the 3 processors is being used, with no timing dependencies.

Jon Shiell

users386/other #18, from skluger, Fri Dec 19 11:09:55 1986. A comment to message 17.

Just what sort of instruction are you thinking of? Must be some indirect immediate 16-bit move with lotsa segment override prefixes...

users386/other #19, from intel, Tue Dec 23 12:54:13 1986. A comment to message 18.

I am afraid I can not recommend using the maximum instruction size as valid means of determining the processor type. Tom Pennello (metaware) has previously posted the "official" method of determining the processor type. While it is more than two instructions it is much safer because of differences between stepings than your method

users386/other #20, from villi, Mon Dec 29 10:03:52 1986. A comment to message 19.

Example: REP REP REP LOCK LOCK LOCK REP REP REP STOSB SS:

Big Question: Where do all the electrons from the LOCKs go? (Reference: Skluger's observation re LOCK pin on IBM PC AT...)

WHATEVER ARE YOU DOING?

users386/other #21, from curtf, Fri Jan 16 09:10:36 1987.

I'd like to ask a question of those who own 80386 machines:

What are you doing with them?

I mean, since there is precious little 386-specific software out right now, what are you using the speed and power for? Are you busy developing software? Do you have a number-crunching application where the 286 is just too slow? Were you also the first on your block to buy a VCR?

I've used 386 machines, and they are certainly wonderful, but I am truly curious to know why people are actually *buying* them now.

Curt Franklin Co-moderator

users386/other #22, from mced, Fri Jan 16 09:56:55 1987. A comment to message 21.

Well, at the moment I'm BIXing on mine. It is a swell tool for anyone doing serious 'C' development - compile times shrivel up and evaporate. They at least get accelerated to the point that you don't forget what changes you made by the time the compile is done.

In addition, I'm doing some 80386-specific development on tools and environments to support software development projects, particularly ones spanning several machines and programmers.

users386/other #24, from csmedley, Mon Jan 19 08:38:38 1987. A comment to message 21.

We are using the '386 to explore running multi-user applications. The power of 286's for multi-user leaves something to be desired, as does the 386 with the existing software. I am hoping for a big improvement in the speed with 32 bit software.

users386/other #25, from viewlogic, Thu Jan 22 20:35:26 1987. A comment to message 21.

We here at Viewlogic now have a number of Compaq 386's, and use them mainly to demonstrate our PC-based CAE package. A significant number of our customers are buying these machines, and we are trying to make sure that our software doesn't have any problem with them. We are, by the way, running plain old DOS on them. In the future, we will no doubt be trying out new OS software when it becomes available, and possibly be writing software for it, but not for now.

Alan Medsker (at Viewlogic)

continued

DESPERATELY SEEKING UNIX...

os386/unix #2, from geberhardt, Thu Nov 20 09:55:15 1986.
I am looking for a UNIX BOX running 5.2 or 5.3, native on a 386. One of my friends says that the cdb debugger has been ported to such a system but cannot identify the manufacturer. All leads appreciated.

George.

---

os386/unix #4, from curtf, Mon Nov 24 15:15:30 1986. A comment to message 2.

At COMDEX I saw an 80386 UNIX box, made by SCI, a company out of Huntsville, Alabama. Try looking in the comdex.fall conference for my report on the machine.

Curt Franklin

---

os386/unix #5, from tpennello, Tue Dec 2 05:08:05 1986. A comment to message 2.

call peter rowell at 3rd eye software, 415-321-0967, for cdb for unix 386 tom pennello, metaware

MULTITASKING SPECIFICS

os386/unix #6, from fheilbronner, Mon Jan 5 17:00:28 1987.

What kind of degredation can be expected on a Compaq-386 running Xenix/286 if I set the Compaq up with 5MB of their 32bit hi-speed static RAM. Can I hang 6 or 8 cpu intensive tasks on the box without having to send out for dinner? Are their any other *NIX flavors other than Xenix running on 386, now or in the near future?
Jim Pauley = NOT(fheilbronner)

---

os386/unix #7, from bwong, Mon Jan 5 23:38:12 1987. A comment to message 6.

Microport Systems either has released or is about to release their System V.2 (3?) for the 386. I haven't heard anything about this, except that they say that it's around. I do know that their 286 version is very disk-intensive (much more so than Xenix -- we know, we have both), and performance improves *VASTLY* when you give enough (1mb per user) memory. Once I/O goes to the floor, Microport is quite fast. I'd expect their 386 product to behave similarly. .s oops

---

os386/unix #8, from tpennello, Tue Jan 6 01:20:33 1987. A comment to message 6.

I suppose there aren't many comments because there aren't many unixes out there. Interactive's system V port is in beta and being tested by at&t now. I have a copy and it seems to work; we brought up our C & Pascal compilers on it.

---

os386/msdos #1, from curtf, Mon Nov 3 10:43:57

The 80386 has joined the long line of Intel chips to run under MS-DOS. While the world anxiously awaits the introduction of Version 5, many 386's are shipping with DOS 3.2. How is running DOS on the 386 different than running it on an AT? on a PC? Are you satisfied with the performance of the 386 under DOS? Have you run any benchmarks? This is the place for your questions and comments on running MS-DOS on the 80386.

Curt

WHICH OS WOULD YOU CHOOSE?

os386/msdos #2, from img, Mon Nov 17 10:03:22 1986.

I'd like to ask the folks out there, "Which of the 386 operating environments do you like, and why? Which one would you buy for your own 386 based machine? Choices seem to be among straight MS-DOS, MS-DOS under UNIX or Xenix, or one of the new multitasking VM environments.

---

os386/msdos #4, from dondumitru, Mon Nov 17 19:14:10 1986. A comment to message 2.

I tend toward wanting MS-DOS under Unix, or equivilent. "Maximum utility" and all that. When the software becomes available for a "standard" 386/VM environment, that will be *nice*. (But it will take some time.)
Donald

---

os386/msdos #5, from dtuttle, Mon Nov 17 23:27:27 1986. A comment to message 4.

The choice of operating system clearly depends on what you want to do with the system. For general software development, the choice would normally be one of the Unix-philosophy systems. Add the MS-DOS virtual en-vironment if you want to run or develop MS-DOS programs.

If you want a general purpose computing engine, try the multi-tasking VM-style system. I am an experienced advocate of virtual machines, being one of the authors of IBM's VM/370 Control Program (rel. 1,2,3) and a devotee of multi-tasks for a single user.

---

os386/msdos #6, from geary, Tue Nov 18 04:47:27 1986. A comment to message 2.

The 386 environment I want is 386 Windows. Unfortunately, it's pretty questionable when it will be available.

---

os386/msdos #7, from greenber, Tue Nov 18 09:21:30 1986. A comment to message 5.

Want a decent operating *and* MS_DOS?? I hear Micro-port SYS V/AT should be available for the 386 beast soon. Then you simply use MERGE386 by LOCUS and voila! You found a way to cripple your UNIX machine, but now you can run lotus! :-) ross

---

os386/msdos #10, from paul.hoffman, Tue Nov 18 22:23:28 1986. A comment to message 7.

Having worked on a contract at Locus (not the Merge project), I can say with all certainty that using Merge386 will ***not*** cripple it. It is a seamless drop-in that is executed wonderfully, even for those of us who hate Unix (and use it all too often).

MS-DOS COMPATIBILITY IN 386 MODE

os386/msdos #13, from villi, Mon Nov 24 20:31:36 1986.

It looks as if we are going to need two kinds of operating systems for the 386. One kind is the "user" operating system, which is probably going to be good ol' MS-DOS 3.x running in Virtual 86 mode. The other kind is the "system" operating system (Yeech!) which "sits under" all the concurrent MS-DOS sessions and manages common resources.

As we all know, there are some products on their way in this category. However, you are (as I understand it) still left with limitations like 640K (or 1 Mb) of RAM for each Virtual 86 session. The trouble comes when I want to write application software that really USES the 386, multi-megabytes and all. This software would have to be initiated from Virtual 86 mode under MS-DOS and run under the SYSOS, returning to the MS-DOS session when finished. SYSOS must therefore be able to provide my program with all kinds of OS services, just like MS-DOS can.

I am interested in knowing what the 386 SYSOS'es under development provide in this respect. Also, we have another problem. My application is probably going to be developed under MS-DOS, at least to begin with. That means that I need development tools that can produce code for such a "mixed" MS-DOS/SYSOS environment. Segment registers ain't what they used to be, which means that for example good ol' Lattice C (large model) and other C compilers are useless; the pointer arithmetic no longer works. The linker is also not up to the task. Are those problems being solved by Microsoft in their Advanced DOS? I really hope so, because what application developers need is a TOTAL solution. This may also explain why IBM and Microsoft took so long to develop a 286/386 OS. You really have to write the whole gamut from scratch: linkers, compilers, assemblers, etc. (Unless you use Xenix, but that is not the issue here.) And anyway, we WILL have to rewrite parts of our applications to run under ADOS, if they ever make use of the segment architecture of the 8086.

The lesson to be learned from all this: the 386 SYSOS'es being developed and "introduced 1Q87" are probably not what the software developer needs. They may be nice for the user who wants to juggle his existing 640K MS-DOS 3.x applications around, but they do not end the wait of those who want to really USE the 386 properly.

---

os386/msdos #14, from jshiell, Mon Nov 24 22:19:48 1986. A comment to message 13.

Phar Lap has an assembler/linker/runtime system for protected mode that is real (I have used it!!) and Medowware has a protected mode "C" compiler that can run in the Phar Lap system (I have not used it but I know people who have). The folk s from Phar Lap (Richard Smith et al.) are on BIX.

Jon Shiell

---

os386/msdos #16, from curtf., Tue Nov 25 16:38:16 1986. A comment to message 14.

Perhaps this is the place to begin putting together a list of compilers that take advantage of the 80386's protective mode. We heard about several "We'll be shipping RSN" products at COMDEX, but we'd like to hear about anyone actually using these products. Put your lists here...if we get enough products we'll put them together in the "digest" topic. Thanks, Curt Franklin, co-moderator

---

os386/msdos #29, from tpennello, Tue Dec 2 04:51:00 1986. A comment to message 16.

MetaWare is currently shipping its High C and Professional Pascal compilers that make use of the 80386 protected mode in conjunction with Phar Lap's DOS extender.

Most of our customers want to look at C rather than Pascal. Phar Lap has already used High C 386 to rehost its 386LINK linker from Microsoft C (286) and has obtained a 25% increase in execution speed and a 20% reduction in object size. Expect in general to obtain some object code reduction but not always a great speedup.

Once you go to 32-bit mode you pay for 32-bit offsets. Two programs I wrote for personal use didn't run much faster (perhaps 5%) in 386 protected mode vs. 286 real mode, although they were smaller. The main speed win from 286 mode to 386 protected mode occurs if you make considerable use of 32-bit arithmetic, especially multiplication and division, which is slower in 286 real mode. [These product announcements were in response to your request for product info] Tom Pennello, MetaWare

---

os386/msdos #17, from tanj, Tue Nov 25 19:04:40 1986. A comment to message 13.

Why is Xenix/Unix "not the issue here" ? It exists now on machines both more and less powerful than the 386, is doubtless in a fairly advanced state in Intel labs (maybe released, I'm not quite up to date), and has all the tools you mentioned. With 386/Merge or equivalent software you can launch Unix applications from an MSDOS shell if you so require, since you have access to Unix files, named pipes, and maybe ordinary pipes. If Locus are one of the Beta sites for DOS 5 you could bet that there are other ways to communicate between DOS and UNIX processes.

UNIX has its imperfections (argh, not rwars please!) but most of the horror stories relate to years past when it was the only decent OS that permitted itself to be crammed into too small a machine. It is now a plain vanilla OS with proven versatility, so any other host for DOS partitions looks a real long shot.

---

os386/msdos #18, from tanj, Tue Nov 25 19:04:56 1986. A comment to message 13.

A little care with terminology is worthwhile. Protected mode is the 286 mode designed in 1980 as the micro answer to the PDP 11/70. The nicest feature of the 386 is we can forget about it. Protection exists, but it is no longer obtrusive and a more appropriate term for 386 programming is paged virtual mode.

It is most likely that a typical application runs with one 32 bit segment and the OS will use page mapping to place the code, data, and stack in widely separated parts of the address space, like any other 32 bit machine. The two segment style - code and data/stack - looks like a "natural" fit but it forces long (48 bit) pointers for constants in the code segment which is just ridiculous for 99.9% of applications.

Existing 32 bit machines are normally run using the paging to control separate (shared) code, data and stack sections of one address space so the design problems are thoroughly understood. Some OS's, like Unix 5.2, allow the code area to be sub-divided into page mapped "segments" for dynamically linked libraries, which means that each application does not need to carry around a copy of the run-time library.

I think you will find the main use for segments on the 386, at least until someone is seriously exceeding 32 bit address spaces on one process, will be for hardware assisted gates between tasks, into interrupt handlers, and into virtual 8086 mode. You need fancy compilers if you want to write the OS, but for applications you can just relax and forget the segmentation. I think in the long run the 80286 form of segmentation will be a dead end for DOS-oriented programmers, since DOS 5 (which is apparently protected mode, not paged mode) will look rather painful by comparison.

---

os386/msdos #21, from jshiell, Wed Nov 26 13:42:40 1986. A comment to message 18.

The other problem with Segmentation is that it is SLOWER then using a flat space, and because there is only one page table there is no additional virtual space available via the use of segmentation.

Jon Shiell

---

os386/msdos #22, from tanj, Thu Nov 27 20:21:47 1986. A comment to message 21.

You are right it is slower, but what do you mean about only one page table ?

The fundamental reason it is slow is that loading a full pointer requires 14 bytes to be fetched from memory (2 for selector, 4 for offset, 8 for segment descriptor). Even the 386 slows down for that. There is no practical limit to the number of page tables, except for RAM. Each task has a different base pointer (CR3) which selects a page directory, which in turn points to up to 1024 page tables. A minimum process needs 4 of 4k pages resident just to hold the directory and 3 page tables (code, data, stack), which must be resident and are one of the disadvantages of the architecture. However it is possible to limit the number of pages of resident RAM by keeping only a high-priority set of tasks in real tables and copying table entries back and forth as processes change priority, since a typical task only has a few tens of 32-bit descriptors in use. I would expect to see from 128k to 256k bytes allocated to page tables in a 4M byte (36 RAM chip) work station.

Even with 16kb of overheads per active task it is a bargain if one can use 32-bit "small" model. Large model on the 80386 will be slightly more efficient than on the 80286 because there are 2 new segment registers which allow a decent compiler to keep a few "register" class pointers, but in view of the typical 25% difference between small and large code on the 8088 it would be optimistic to expect less than 15% difference on the 386. In other words, if you use compilers and your application has more than 100k of object code then non-segmented mode is not only faster but also on balance smaller, even with the extra system tables. And that is before you count the savings in data space due to smaller pointers!

It is interesting that the programmer of the 80386 in paged mode will see very little difference between from using the 68020. NS32032, VAX, or any other modern

continued

32 bit device. Assembly code will decline (but not vanish), the screen will be windowed and bit-mapped, and the OS will be (at least modelled upon) Unix and protected.

os386/msdos #23, from jshiell, Sat Nov 29 00:12:57 1986. A comment to message 22.

Only one page table may be active at a time and all active segments must map into that table when paging is active. Jon Shiell

os386/msdos #24, from tanj, Sun Nov 30 06:01:53 1986. A comment to message 23.

Ahhh..the light dawns ! So the only way to make use of the 64 trillion byte (46-bit) virtual address space (32 trillion in user-space) is to use multiple processes, each of which might just as well be in 32-bit "small" mode.

There are days when I yearn for the clean instruction set and architecture of a 68020, or 32x32, or Clipper, or VAX, or... Instead, I will doubtless use an 80386 which is a stretched version of a widened 8080. It is for the outside chance of escape that I keep looking to UNIX.

os386/msdos #25, from geberhardt, Sun Nov 30 13:01:57 1986. A comment to message 24.

But 32 bit address space in small model is the same as is currently available on the 68020. Actually I am in the process of getting a 386 C Compiler up, and I expect that A 3 segment version would be much easier to set up.

The segemnts would be 1 for code, one for data and 1 for stack. It may be nice to have the data and stack segments in the same physical address space, but enve then it would be 2 times the size of the 68020.

Still the real limit will be the size of disk for swap purposes. My biggest machine has only 400Mb of disk, and that cost $17000 for the disk alone.

George.

os386/msdos #33, from tanj, Wed Dec 3 20:00:55 1986. A comment to message 25.

You don't need segmentation to implement separate address spaces and grow stacks. What you do is use the most significant bits of the address space as "segment" bits, as for example with the IBM RT, and then the paging hardware keeps track of what's what. For example, you could grow the stack down from 4 Gbytes, the data up from 1 Gbyte, use the first 256 Mbytes for application software, and the next 768 M for system software and HLL support libraries. To grow your stack the OS knows that if you try to use the next page down it should allocate another page, fix the page descriptor, and restart your program. You do have the disadvantage that the protection system is weaker since you can't catch a bad data pointer which scribbles on stack (for example), but I'd rather live with that than the horrors of 48 bit pointers on a 32 bit machine. Would you like to try that in C (or any other practical HLL) ? The funny segment registers will basically support 286 emulation and special tricks, hand coded, in the OS. In a few years time it will seem a shame to be paying for the unused silicon and heat dissipation they represent. I can't even see the 486 changing that, even if it does allow more than 32 bits of paged address, since there are so few applications that need more address space enough to accept 48 bit pointers (with 112 bits per segment load).

os386/msdos #27, from jshiell, Mon Dec 1 01:02:45 1986. A comment to message 25.

There is not point in using segmentation in the 80386 unless you don't use paging because the page table can only describe 2**32 bytes (also see 386users/flames), ie. the page table can only map a single segment (at the segments maximum size 2**32 bytes).

Jon Shiell

os386/msdos #30, from tpennello, Tue Dec 2 05:10:36 1986. A comment to message 25.

No, you don't want a separate segment for the stack. otherwise, you couldn't use simple 32-bit addresses to reference any data item in C. E.g., the following couldn't work in your model without 48-bit pointers: sub() { static int x; int y; f(&x); f(&y); } A 32-bit reference to the address of x or y is insufficient for the receiving routine (f), since it doesn't know which segment register to use Tom Pennello

os386/msdos #31, from jshiell, Tue Dec 2 11:04:49 1986. A comment to message 30.

Unless a totally flat model is used (ie the SEG regs. are NEVER changed). Jon Shiell

os386/msdos #32, from tpennello, Tue Dec 2 16:19:27 1986. A comment to message 31.

Even if the seg regs are not changed, if ds <> ss then 32 bit addresses still don't cut it. Do you define "totally flat" as ds=ss=cs=fs=gs=es?

COMPILERS FOR 386

os386/msdos #36, from intel, Fri Dec 5 03:25:42

In a previous message someone asks for a list of 386 compilers. There are number of companies doing 386 compilers. Most are in the beta site stage for their compilers.

		Host
Franz	Lisp	Unix System V.3/386
GreenHills	C, Fortran, Pascal (later)	Unix System V.3/386
Lucid	Lisp	Unix 386 and ?
Metaware	C, Pascal	DOS, Unix 386 others
LPI	C, Fortran, Pascal, Cobol etc	Unix #86
Ryan-MacFarland	FORTRAN, Cobol	Xenix, Unix and ?
Silicon Valley Software	C, Fortran, Pascal	Unix V/386
Intel	C, ASM	Xenix 286, DOS VAX/VMS (Q1)

I know I've left some out these are the one I have been dealing with and which we have done press releases with.

I am very familiar with two companies compilers GreenHill's and MetaWares. GreenHills is most famous for their 68K compilers, but recognized a winner like the 386, they are currently make a real nice optimizing compiler. It generates amazing Dhyrstone numbers (in excess of 6200 (version 1.1) and seems to be remarkably free of bugs. Metaware's compilers are used by big-name software developers in the MS-DOS world. Their compiler for the 386 also generates superior code and seems very stable. Whats real slick about the MetaWare compiler is you can use it with Pharlap's 386linker and DOS Extender product to generate real 32-bit code and run it on your 386 PC, today.

Clif Purkiser

COMPILERS?

os386/msdos #38, from jwvincent, Tue Dec 9 08:19:28 1986.

I am interested in porting a manufacturing application that currently runs on Apollo, VAX, Prime, and HP320 systems to a Compaq386 or Zenith386. This program's object code is currently 3meg and growing. I am *not* interested in segmentation! I would like to operate under MSDOS since it is the std OS at this time. The program is in very straight FORTRAN 77. Questions:

What would be the best compiler for this task?

Is there any prospect of a vertual memory OS coming to my rescue?

What can I do to get around the short file name problem?
(Mfg'sr like to ref things by part numbers longer than 6 chars)
Please help.
Thanks, JWV

os386/msdos #39, from feenberg, Tue Dec 9 18:18:41 1986. A comment to message 38.

If you join the FORTRAN conference you will find that the favorite compiler of many is the LAHEY F77L compiler. I suppose 3meg of object files is about 300,000 lines of source code. Programs that big have been converted to run on PC's, but it can take a few days! The current crop of PC FORTRAN compilers art as good as the best mainframe compilers.

os386/msdos #42, from jwvincent, Wed Dec 10 08:18:07 1986. A comment to message 39.

You have given us credit for slightly tighter code than actual. The source is over 150,000 lines though. The real issue isn't having to segment the code once and have it work, instead it's that the program is moving ahead with many new features and enhancements which must be available on all hardware platforms with a minimum of effort. We currently have the program gene ric enough to move between 32 bit systems quite quickly. We don't want to lose this flexability by supporting a PC.
No prob' neether kin eye!

OP SYSTEMS FOR 386

os386/msdos #46, from gtromble, Sat Dec 13 15:03:00 1986.

Today who can shed some light on the 386 multiuser op sys being marketed by Software Link. I'm sure it's probably a revision of their multi-link software and not a true 386 op sys. Any beta testers out there?
Galen Tromble

os386/msdos #48, from donbrady, Sat Dec 13 21:38:21 1986. A comment to message 46.

I had a demo of the 386 MOS by Multilink and was NOT impressed. I crashed it 3 times just running their demos of Lotus, Wordperfect etc. on a three- terminal system. Also their feature list is pretty strange and sort of beside-the-point. I.e. they mention minor operating environment commands (such as the "NO" prefix) which are have nothing to do with 386-mode operation. Of course that was a pre-release version but it seems to me they have a LONG way to go. Yes they claim it is a new operating system but it LOOKS more like a Multilink extended to support virtual 8086 mode. There are some nice task-switching features etc.

os386/msdos #51, from csmedley, Tue Dec 16 09:36:44 1986. A comment to message 46.

Ihave the BRIDGE version of the software, which is MUltilink with a protected mode driver. However I have also seen the pre-release versions of PC MOS 386 This appears to be a totally different animal down to displayinmg a directory with the period symbol delimiting the extensions and other strange things. These differences may change in the final release due Feb 28,1986.

os386/msdos #49, from billn, Sun Dec 14 12:01:51 1986. A comment to message 48.

In theory it looks like it *could* be good (someday). The demo at Comdex with a 386 system running two terminals was so slow as to be unusable, but that could be from many causes, probably fixable.

As a broader concern, there is really a problem in the multiuser on an AT or 386 area. No single system has a dominant position causing scattered efforts in the applications area. This is a chicken & egg problem I do not see being solved very soon. Unfortunately. What is *really* needed are development tools that can span the many multiuser OS environments with minimal effort by developers. This could finesse the OS problem, which is really an application problem (not enough applications, leaving MS-DOS dominant).

Any ideas? BillN

os386/msdos #50, from jshiell, Tue Dec 16 02:05:51 1986. A comment to message 49.

Applications developed without useing the seg. regs. (ie CS = ??, SS = ??, DS = ES = FS = GS) should run in most enviroments. Both 386/ASM from Pharlap and HIGH-C from Metaware (Hello Richard & Tom) support this type of seg. structure. Maybe someone who knows the DOS under Unix OS. could state if they are also using this model. Jon Shiell

os386/msdos #52, from tpennello, Wed Dec 17 03:36:15 1986. A comment to message 50.

John has asked me to summarize current seg reg usage of the High C compiler. This is actually dictated by the Phar Lap environment, and also by architectural constraints of small model. Here is the seg reg arrangement:

```
SS = DS = ES = FS = GS
    CS = ??
```

This is also true of Intel's system V unix, release 3.0. That SS = DS is DEMANDED by small model; you can't get away without it. Let's put this one to rest:
`int x;          main() { int y; f(&x,&y); }` The address of x and of y is a 32-bit quantity. In function f, the dereference of the two passed-in pointers are made relative to DS:

```
        f(int *x, int*y) {
            *x = *y;      /* mov eax,x; mov eax,[eax];
                           mov edx,y; mov [edx],eax */
}
```

Here the x and y pointers are referenced off of ds in the two mov instructions. If SS were not = DS, the address of y passed in would be invalid.

Now, whether ES = FS = GS and/or whether ES = DS is a separate decision. In Phar Lap's environment, DS=ES=FS=GS=SS. Under UNIX ES=DS=SS, but I don't reecall if FS and GS are set to anything in particular.

os386/msdos #53, from tanj., Sun Dec 21 04:49:40 1986. A comment to message 52.

One thing puzzles me. You say that CS is unconstrained. What kind of pointer do you use for parameter-functions ? Don't you put initialised const in the code segment (where code is shared, as with System V), and how would you handle something like an implementation of the regcmp() functions which places code into a heap-block and then tries to jump to it via pointer ? It looks like you either accept 48-bit pointers or place restrictions on the programmer. I can't see what is gained by leaving CS open, seems easy enough just to tie it CS=DS=SS.

os386/msdos #61, from tpennello, Sat Jan 3 00:34:04 1987. A comment to message 53.

Sorry to take so long to reply; just got back from two weeks vacation. CS can be unconstrained because the procedure call instruction, for short calls, implicitly uses CS. So 32 bits suffices. But note that dereferencing a function pointer as data -- e.g., casting it to (int*) and dereferencing -- does not produce a value in the code area, since the deref uses DS implicitly.

continued

If you allow CS = DS you eliminate the read-only protection given to code. Data references can then clobber the code area. No reason to allow this unless someone really wants it.

---

os386/msdos #66, from tanj, Sun Jan 4 04:32:37 1987. A comment to message 61.

CS=DS does not lose read-only protection of code, if the OS is designed to map code and data into distantly separated pages and the code pages are read-only. That is a normal practice in paged virtual OS's (viz. 370 VS/XA, VAX/VMS, BSD 4.2, System V.2.2 and later).

---

os386/msdos #68, from tpennello, Tue Jan 6 01:25:52 1987. A comment to message 66.

Yes, if you use paging. You need not use paging, right? I don't recall if the Phar Lap DOS Extender currently uses paging. Richard Smith, can you hear me? According to friends at INtel, embedded applications people may want to use 48-bit pointers to get very local protection. Quite a few of the bugs I had to get out of our C & Pascal ports to 386 system V unix were hard to find since we overran an array and the symptom didn't show up until later. On a 286 system we would have been caught earlier, frequently at the array overrun, since sometimes an array has a segment of its own. Hence we've kept intact the medium-compact-big-large memory models of our compilers so we can support 48-bit pointers.

---

os386/msdos #69, from tanj, Tue Jan 6 18:43:40 1987. A comment to message 68.

I put in a message earlier analysing the pros and cons of paging, and why I think it is the right 386 choice. Maybe it was in the vm topic. I've used the 86 family since 1980. I've written an object oriented OS, and I've used Intel's RMX. They have their advantages, but in a competitive commodity market (which is what the 386 is about: if you want just nice 32 bit micros Motorola is way ahead, but the 386 will inherit the business PC market) after 6 years with the beasts I just don't think strict object oriented code slavishly converted into Intel's idea of a segment is the way to go. You said it yourself: "sometimes an array has a segment". Sometimes, not always. The reason is that your compiler would crawl if you always did it and no-one would buy. If you move to a 32 bit environment where the simplest way to implement malloc() is linearly then "sometimes" becomes "rarely". You will design your compiler to allow people to set a high threshold on the size of arrays given their own segments or just turn it off, since they will want their code to be small and fast.

In some languages array access is more recognisable than in C so a bounds-check is more reliable. Anyone writing serious embedded code should look for a good Pascal or Ada compiler, as those languages help you write reliably, at the expense of a few favourite tricks. Only the arrays need bounds checks, which are selective and cost less time than condemning the whole application to loading 48 bit pointers with descriptors.

I do expect that whichever 386 OS wins out will support both styles, and have paging. For the majority of applications the paging is by far the more valuable feature and programming in a linear model is simpler and will give the best running speed. Some special applications will run in segmented object-oriented mode, and those may include programs under test up to the Beta phase.

---

os386/msdos #70, from tpennello, Wed Jan 7 05:36:57 1987. A comment to message 69.

r.e. "array has a segment of its own". Whether or not it does doesn't affect speed of the compiler on the 286; in the particular application, they were all pointers to arrays, so a 32-bit pointer load was necessary always. Sometimes, due to heap management, the segment in which an array was allocated was not shared with anyone else, and provided for array bounds detection, since the bounds of the segment were close (possibly identical) to that of the array.

---

os386/msdos #71, from tanj, Thu Jan 8 17:45:13 1987. A comment to message 70.

Now write that paragraph again, but for a 386... 32 bits changes the compiler and allocator strategy. PS. if a 386 OS does not provide paging then why would it win on the competitive market ? Considering how nice it is on existing workstations, why shouldn't PC usersfind paged virtual addictive ?

SEGMENTATION

---

os386/msdos #56, from dtuttle, Tue Dec 30 23:35:08 1986.

It's really interesting to observe ("listen to") the controversy on segmentation versus large, linear address spaces. There are both benefits and drawbacks for each approach, and a significant amount of "religion" (as in unsupported beliefs) behind any choice of one or the other.

The "original" memory management schemes, as much as there is an original, were based on what is now called bank-swapping, a la Intel/ EMS (I think). <ref: 1961-68, MIT Compatible Time Sharing System on an IBM 7094 with two banks of 64K each, 36-bit words> Around the same era Burroughs had some sort of virtual memory scheme, but I've never found out much about it. Significantly, the CTSS/7094 system used the bank-switching for memory protection as much as it did for the extra space. The operating system ran in one bank, user programs in the other. This also solved the address range problem, since a user program had access to a full 64K range of its own.

Next into the fray was (I think) the IBM System/360 Model 67, with the first large-scale implementation of dynamic address translation and demand paging. <ref: 1968 IBM Time Sharing System/360 (TSS/360); 1969 IBM CP-67/CMS; 1971 (?) Michigan Time Sharing (MTS)> The S/360-67 had your choice of 24-bit or 32-bit virtual addressing, on a physical base of 24-bit addressing (1-Mbyte segments, 4-Kbyte pages). Significantly, the System/360 machines had a storage protection scheme which was com- pletely independent of the virtual memory/address translation hardware. Conventional operating systems such as OS/360 and TSS/360 used storage keys to protect the system code from the user code, and used the memory management facilities to isolate users from each other. Virtual machine systems such as CP-67/CMS and MTS used the virtual memory facilities to isolate everything (incidentally allowing "user" programs the full use of the storage protection key mechanism).

The IBM System/370 Virtual Storage announcement (1972) brought in the notion of choosing between 1Mbyte and 64Kbyte segments, and between 4Kbyte and 2Kbyte pages. The much later Extended Architecture facility rediscovered the notion of system space vs. user space (see 1961, above) along with the notion of 31-bit addressing.

While this was going on, somewhere in the early 1970's, DEC discovered that 64K bytes was not a lot of memory for a multi-user machine and started using segment registers and memory mapping. The real need was for a way to utilize a physical memory space which was larger than the maximum logical address, but they included support for variable-size segments and segment-based protection mechanisms. This was, I think, the first explicit combination of memory management and memory protection schemes. Thus it was possible to control access to areas of memory with- out an expensive, physical-memory-based, storage protection scheme.

The Intel 8086/80186/80286 segmenting most likely grew out of the same basic need -- how do you address more than 64K bytes when you're stuck with a 16-bit logical address? -- but that's by far not the only reason for continuing with it. The combination of per-segment attributes such as read/write, read-only, execute-only, "gate segment", etc., with the basic virtual memory functions, results in a very powerful "object- style" capability for multi-tasking and/or multi-user environments. — Dave Tuttle (currently: Principal Technical Consultant, Prime Computer Inc.) (formerly: VM/370 Control Program System Architect, IBM Corp.)

---

os386/msdos #57, from wmiller, Wed Dec 31 20:35:34 1986. A comment to message 56.

DEC? I thought it was the Multics work on GE 635/645s in the late 60's and early 70's that pioneered the kind of segmentation, protection rings, etc. that are now appearing in the 386.

---

os386/msdos #58, from tanj, Thu Jan 1 06:00:36 1987. A comment to message 56.

If memory serves me correctly the Ferranti Atlas had paged virtual memory around 1963. It was a fast machine for its day and derived from work at Manchester University. The last one was switched off in the late 70s. Ferranti's computing interests were one of the components which merged to form ICL, and there have always been virtual storage machines in the ICL product line.

I read somewhere that the virtual memory scheme in the SCS machines (which Burroughs bought) was also inspired by the Manchester ideas. I have worked on the 6500, which is both paged virtual and segment-protected, with HLL/Algol stack-oriented instructions. I don't know how much the Californian SCS invented independently, but that whole ball of wax was what Manchester promoted. The key innovation which IBM can lay claim to in its VS design is the virtual machine, with the hyperviser able to simulate not just separate memory spaces but a whole private configuration of peripherals. Object-style programming does not necessarily involve overheads of object (segment) descriptors at run-time. It is true that such devices can help with catching certain kinds of errors, but if a program is error-free then all that checking is just parasitic. You can get the same kinds of checking with compile-time and run- time software checking during development.

History suggests that only specific markets will pay for paranoia overheads in hardware. All that is necessary is the separation of processes so that one can crash without affecting others, and paging with protection does that job quite well.

Gate-segments are just a variation on inter-process or inter-virtual- machine calls, which can be done otherwise. I don't have anything per se against segmentation. However, the style of segmentation promoted by Intel for the 80286, and taken to extremes with the iAPX-432, causes run-time overheads due to descriptor loading. In a 32-bit machine where all the key virtual and protection facilities are available in the paging hardware, I think that kind of segmentation is irrelevant to the majority of applications. The 80386 is best used linearly.

---

os386/msdos #59, from dtuttle, Thu Jan 1 15:23:22 1987. A comment to message 57.

The MIT/GE Multics project was the pioneer in terms of ring-based protection, but I don't know where the segmentation approach orginally came from. The Multics work was also the basis for the current Prime 50-series ring-based architecture. As far as I know, the DEC memory management (PDP-11) was at least an early and aggressive use of logical addressing as a means of utilizing a large physical memory.

There were at least two major trends of development of the virtual memory and memory management capabilities -- from the high-end down and from the low-end up. Concepts pioneered on large-scale systems have been adapted and utilized on smaller and smaller systems, and (because of the compatibility requirement) architectures designed for very small systems have been appearing on larger and more powerful machines.

The current architecture of the Intel 80386 is an interesting mix of small-scale features inherited in the name of compatibility and some very sophisticated additions which are justifiable only for multi-user large-scale systems.

---

os386/msdos #60, from dtuttle, Thu Jan 1 15:35:33 1987. A comment to message 58.

Interestingly, even the early CP-67/CMS system was not the first virtual machine system. It was based on a research system called CP-40, which ran on a modified System/360 Model 40, with both hard- ware and software developed at the IBM Cambridge Scientific Center in Cambridge, Massachusetts. Earlier yet was M44/44X, based on a modified IBM 7044 and developed at the IBM Yorktown Research Center in Yorktown Heights, New York. I was fortunate enough to be a user of the M44/44X system during a summer job in 1966.

On the other subject, I agree that the "overhead" of segmentation is of no importance to an application -- but it _is_ important to the system. As long as the size of one segment is sufficient, there is no reason for any one _application_ to use other than CS=SS=DS=ES=... If, however, I were writing a multi-user and/or multi-tasking _system_ program or subsystem, I could make very good use of more segments.

---

os386/msdos #62, from dathomas, Sat Jan 3 13:23:15 1987. A comment to message 58.

Nice to see that someone does the homework/history. The Atlas was the first machine. The other interesting machine to use paging was the SDS then Xerox Data Systems Sigma series. We ran our university computing on a 64K 32 bit word Sigma 7 with 8-16 users and 2 batch streams in 1970! The machine had a head per track swapping disk with a 4 ms latency and 3.5 MB transfer rate! The machine also had multiple registers sets 16, for fast context switching. The CP-V OS was ported to HIS hardware when Xerox exited the computer business in 1975. The Vax OS internals have a familar similarity. Signed An Old Systems Programmer

Lets See 16 users on an 836! <grin>

---

os386/msdos #63, from billn, Sat Jan 3 13:31:05 1987. A comment to message 62.

Actually, given the right supporting hardware and sufficient memory (say 8 meg), it might not be too bad. But you will need a *good* OS, much better than currently available.

BTW, as I recollects, the Atlas, a British machine, was circa 1959! BillN

---

os386/msdos #65, from tanj, Sun Jan 4 04:32:34 1987. A comment to message 60.

A multi-user/multi-tasking system needs multiple page tables. These may give each process a combination of private pages, shared read- only code, and shared inter-process variables. The only thing segmentation buys you (on the 386 at least) is exact control of the length of each area: cute, but not worth a performance loss.

---

os386/msdos #67, from billn, Sun Jan 4 11:47:06 1987. A comment to message 65.

Or you can share on a segment basis which sometimes makes more sense. BillN

---

os386/msdos #72, from dtuttle, Fri Jan 9 00:11:10 1987. A comment to message 65.

I am getting a little tired hearing about "performance problems" on a machine which is capable of 3.2 to 3.8 MILLION instructions / second. If there are any performance problems, they are not the result of the architecture of the processor -- they are the fault of the surrounding system. Put a 16-MHz 80386 in a reasonable package (64Kbyte cache, DMA multi-channel, overlapped disk I/O, (semi-)intelligent communications controller, etc.) and you have a system which can easily blow the doors off of a medium- to large-sized VAX 8x00 system.

As to 16 users on a 386... Bell Northern Research used to routinely support 1- to 2-second response time for 190-240 users on a System/370 168-AP, which had an instruction speed approximately the same as a 16MHz

continued

80386. The I/O subsystem was not "PC/AT compatible", to be sure, but that it not a characteristic of the CPU architecture.

If anyone actually has a requirement for more than 2**32 bytes of program plus data, or has a program which doesn't run because it has to occasionally use 48-bit pointers, I will happily offer free consulting to work on their problem. It will certainly be worth the time...

os386/msdos #73, from intel, Fri Jan 9 02:37:03 1987. A comment to message 72.

Yes, I agree that it is sort of ludicrous to complain about the performance of the 386. In the right system, it really screams. Even in a system with a poor I/O subsystem like a PC a 386 makes everything appears to happen instantly.

Ignoring if you will my built in bias, I can honestly that speed of the Deskpro 386 at work is enough for me to prefer using to my Mac+ (with hard disk) at home. This is only true because of the speed of the 386. I vastly prefered my Mac 512 to my first IBM PC, and it was a toss-up between my Mac and a 8 MHz PC, but the speed of the Deskpro makes everything seem to happen as soon as I hit the return key.

My 60 page Framework documents load, save, reformat instantly, I never turn the spreadsheets off autorecalc, and even C compiles happen within a few seconds . Eventhough, I perfer the Mac's interface it is frustrated to have to wait for the Mac to perform some CPU intensive operations, like redrawing the desktop.

Clif Purkiser.

WHERE ARE THE 32-BIT EXTENSIONS

os386/expert.query #1, from villi, Mon Nov 3 17:57:09 1986.

The 386 has new 32-bit registers named EAX, EBX, EDI, and so on. Also, I understand that it can do things like MOV AX,[12345678] (that's a 32-bit absolute address). As a compiler developer (who has not received the Intel tech documentation yet), I would like to ask where those features were put in the instruction set. The mod reg r/m structure of the 86/286 appears pretty full. How were the descriptors for the extra registers and addressing modes added? Mustn't the new 32-bit instruction opcodes be at least a byte longer than their 8/16-bit counterparts?

os386/expert.query #3, from skluger, Mon Nov 3 18:15:22 1986. A comment to message 1.

Judging from the manual, they did fit them all in. Try to get the P.R. manual, Order # 230985-001. The ModR/M tables are on pages 17-4 through 17-7. There is something called "SIB" (Scale Index Base) byte which is required for based indexed and scaled indexed 32 bit addressing.

os386/expert.query #4, from mced, Tue Nov 4 00:45:51 1986. A comment to message 3.

As far as the difference between AX and EAX registers, the 386 at all times has a default operand size and a default code pointer size. These defaults are set in the segment descriptor associated with the current code segment.

So if you're executing "32-bit" code, a given opcode will pertain to the EAX register - in a "16-bit" segment the same opcode will mean the AX.

Just as there are segment register override prefixes in previous processors, the 386 has operand and code size overrides: they work as simple toggles to indicate that the next instruction should use whatever size ISN'T the default right now. The code pointer size is used for jumps, calls, and the like.

The SIB byte is a real nice extension. It follows the mod R/M byte in the opcode and its presence is indicated by an encoding in the mod R/M byte. The Programmer's Reference manual is tough to find at present, but it is by far the best reference around. It's the only document I've found which actually EXPLAINS what the 386's swell features are and how to use them. Other writers (especially those of magazine articles) seem to love spouting forth about what CAN be done (copied from Intel spec sheets) but precious few of them seem to have any idea of HOW.

os386/expert.query #5, from villi, Wed Nov 5 16:17:35 1986. A comment to message 4.

Re the default operand size located in the current code segment selector: It is sure gonna be nice to write a compiler which finds out which default operand size yields the smallest code, and then uses that information to optimize. However, I guess the 16-bit mode will be the most popular default mode, judging from experience with common programming constructs. By the way- is there an 8-bit default mode? Of course, I'm joking. Thanks for the information on the Programmer's Reference. It's real gold for me.

MULTITASKING MSDOS

os386/expert.query #2, from villi, Mon Nov 3 18:01:54 1986.

My company is developing an Ada compiler for the 286, running under PC-DOS and MS-DOS. On the 286, we switch the processor into protected mode, do our multi-tasking in hardware (with superb performance), and then use a complex kludge to get back into real mode and DOS (involving a reset and the CMOS reset vectors in the PC AT).

Does the 386 have hardware features to let me do the same thing without kludges? That is, does it allow an MS-DOS task to initiate a program that switches into 386 mode, communicating with MS-DOS while running, and then returning gracefully to the original MS-DOS real mode task when finished?

os386/expert.query #6, from intel, Thu Nov 6 02:04:05 1986. A comment to message 2.

The 386, unlike the 286, can be returned to real mode. This is because the Protection Enable bit in the 386 is not sticky like it was in the 286. In general the way to do this is to reset the PE (bit 0) in the Machine status word which is called CR0 (control Register 0) on the 80386. There are a few cavets when doing this the most important one is to avoid having stale values in the segment descriptor caches. This is done by making a long (intersegment) jump followed by loading the segment register after returning to Real Mode.

Pharlap, who is on this conference (Richard Smith), has a beta site product which already does this. Incidentally, one of the ideas behind the Virtual 8086 Mode of the 386 is to avoid the complexities associated with transistioning between Real and Protected Modes. My recommendation is to investigate having your Ada compiler run in V86 Mode. Clif Purkiser Intel 386 Software Applications manager

386 VS. 286

os386/expert.query #7, from paul.hoffman, Mon Nov 17 21:47:01 1986.

OK, I'm confused. Everyone is talking about how wonderful it will be when Microsoft has their ADOS (MS-DOS v. 5, or whatever) so that we can have real multitasking on the AT and 386 machines. But, as I understand it, the 386 is not really a superset of the 286, and that the protect mode of the 386 is really quite different than the protect mode of the 286. Will a multi-tasking MS-DOS that is designed for the 286 be able to run on the 386? Will it just be a kludge that works but runs slowly? Or will there be a 286DOS and a 386DOS?

Of course, I'm not asking Microsoft to preannounce anything. This is really just a technical question about the differences between the two chips.

---

os386/expert.query #8, from dtuttle, Mon Nov 17 3:31:05 1986. A comment to message 7.

One can make a very educated guess — there will be a 386-only DOS, whether or not there is a comparable 286DOS. The hardware features and capability of the 80386 are too powerful to ignore, especially for any multi-thread system.

---

os386/expert.query #9, from jshiell, Tue Nov 18 00:57:40 1986. A comment to message 7.

The protected mode of the 80386 is a proper superset of the 80286 (UGH !!) thus anything that can run on the 80286 can run on the 80386. Jon Shiell

---

os386/expert.query #10, from villi, Tue Nov 18 01:06:01 1986. A comment to message 9.

Yes, given some previous system register initialization, etc.

---

os386/expert.query #11, from intel, Tue Nov 18 01:25:46 1986. A comment to message 7.

The 386's instruction set is a superset of the 286's. Therefore, 286 OSs will run on the 386. This includes the much-discussed-little-seen ADOS, DOS 5.0, Protected Mode DOS (your favorite term for uSoft's rumored OS goes here).

The interesting question is will a 386 specific DOS developed by one of the smaller companies. (Softguard, Software Link, Pharlap etc) gain popularity before a DOS for the 286 gets introduced.

At Comdex, I saw no less than six new 386 OS which let you run multitasking MS-D OS applications.

In addition to two DOS under Unix products: VP/ix by Interactive Systems, Phoenix Merge386 by Locus Computers

Multiple DOS sessions were demonstrated by Softguard in there VM/386 and by Software Link with PC/MOS 386

Finally, Phar-lap allowed users to run 32-bit DOS applications.

Probably the most impressive product was Convergent Technologies. On their NGEN 386 workstations you could run multiple copies of MS-DOS along with multiple CTOS applications. I was impressed with the speed a the good human interface CT used to switch between applications.

Of the other companies, Softguard's VM/386 and Phoenix/Interactive's VP/ix seeme d to be the closest to being a finished product. Although, I wouldn't look for anybody to have a solid product much before the end of Q1/87.

Clif Purkiser

TASKS

os386/expert.query #15, from johnf, Sun Nov 30 17:47:51 1986.

Where in the course of system software development do you see operating systems for real-time processes becoming available? The two main character- istics I see for such a system are the ability to run tasks at a variety of controllable time intervals, along with a number of othere tasks running as time is available. In addition, there must be facilities for easy and efficient communication of data and semaphores between otherwise independent tasks.

-:- JohnF -:-

---

os386/expert.query #17, from rguilmette, Tue Dec 2 03:27:06 1986. A comment to message 15.

System V has all the primitives you could need for semaphores and message passing, as well as memory sharing. As far as real-time, all you need is to tweek the UNIX scheduler a bit to get that, and many of the UNIX clones on the market have done that (e.g. regulus). See also: "The Fair Share Scheduler" in the Bell Labs Tech Journal, October 1984.

---

os386/expert.query #25, from tanj, Wed Dec 3 20:01:11 1986. A comment to message 15.

There are two kinds of "real time": one is like controlling an aircraft or a chemical plant, the other is like logging instruments or recording voicemail. The first is distinguished by having closed-loop input-output paths with critical timing requirements. Don't use Unix for that, not even a fair-share version, since Unix is not repeatable on fine time-scales. It is also rather larger than is normally required for such systems, which must normally be special boxes with a fixed program repetoire, kept at comms-length from the vaguaries of general time-sharing hosts. If that is what want I suggest you talk to Hunter & Ready, or to Intel (the iRMX group), or to IPI (the MTOS people), and possibly to Alsys (since Ada includes a suitable run-time kernel).

The second class has real-time data throughput requirements, but the rapid response loops are all trivial stuff like handshaking a comms line. Unix can do these fine. You get a version which supports you adding your own device drivers and in your drivers you put all the hand-shaking, and a buffer for I/O data. The data analysis and storage programs operate as normal applications and get scheduled every so often to pick up or deliver data in the buffers. Screen oriented editting is a real-time problem in this class. Not all editors are competently implemented, but Unix can run them with fine hand-eye coordination, which requires loop times well under a tenth of a second. However, they do hiccup occasionally when the system is very loaded or when the editor has not been used for a while and needs to swap back from disk. That is why you need the buffers in the device drivers. Also, don't overload or allow uncontrolled users on machines which must handle real-time work. Within this kind of environment, Unix has the facilities you asked about.

---

os386/expert.query #34, from johnf, Mon Dec 15 03:57:08 1986. A comment to message 25.

It's the first, real time control application that I'm interested in, with other, non-time critical tasks running in a background mode. It is done now, but the advantage of the 386 seems to be that unrelated tasks are easily protected from each other.

-:- JohnF -:-

---

os386/expert.query #35, from tanj, Tue Dec 16 17:50:46 1986. A comment to message 34.

The 286 in protected mode was inspired by KSOS, a military Kernel Secure OS project. Unrelated tasks are protected, so long as the OS design is tight. The 386 is, if anything, a bit more relaxed about security. The NS 32nxx chips have inter-process calls which are effectively switches between virtual machines, and Motorola followed suit with some new instructions in the 68020. I think you will find many 32-bit machines with memory management have similar capabilities of isolating programs from each other, it is one of the normal design goals. If you have reservations about the security of Unix then it won't make a difference which chip you use, except insofar as some machines still run old Unix versions, and there can be differences in the quality/solidity of ports to different chips.

---

os386/expert.query #31, from villi, Tue Dec 9 12:58:28 1986. A comment to message 15.

What you need is an Ada compiler for the 386, regardless of OS. Ada has tasking (real-time), can run tasks at controllable intervals, and has very advanced

continued

intertask rendezvous and communication facilities. (Beware: my company makes Ada compilers.)

os386/expert.query #18, from johnf, Tue Dec 2 04:02:02 1986. A comment to message 17.

It seems to me that it is accepted wisdom that Unix is not suitable for real time control applications, perhaps because of its origins, but maybe because of more concrete reasons, such as the way it would handle analog and discrete (boolean) input and output.

I'll check into Regulus. The article you reference is titled "The Fair Share Scheduler" which puts a *user* rather than real time control emphasis on the system.

Perhaps the problem with Unix is that it is too "big" an operating system to include as a shippable product.

-:- JohnF -:-

os386/expert.query #19, from johnf, Tue Dec 2 04:04:07 1986. A comment to message 17.

One other question: How is the multi-tasking available with System V affected by the architecture of the 80386?

-:- JohnF -:-

os386/expert.query #20, from rguilmette, Tue Dec 2 05:20:19 1986. A comment to message 18.

I don't accept the accepted wisdom on UNIX not being acceptable for real-time. Question authority is my motto. Also, I don't see boolean I/O as being THAT different from character I/O. The Fair Share Scheduler could allow a sort of real-time control in the sense that you can allocate a fixed percentage of available (i.e. non-overhead) cpu time to a given UID (or was it a group of UID's, or was it a process group, oh well it could all be made to work out the same). This percentage would then ABSOLUTELY be made available to that UID, or GID or whatever, whenever needed. That sounds like you could setup one "real-time" UID which would be given a fixed allocation of up to 100% and ta-da! Real-time! Yes? No?

os386/expert.query #23, from patwood, Wed Dec 3 18:56:59 1986. A comment to message 20.

Real-time UNIX...hmmmm Seems to me that AT&T has been using UNIX for quite some time in one of the largest real-time control systems in the world: the telephone switching system, specifically, the number 5 ESS, which is a 3B20 on the inside with DMERT and UNIX running on it. There have in fact been several versions of UNIX specifically designed for real-time applications, including UNIX/RT, which to my knowledge only runs on PDP-11 type machines, and VENIX, from VentureCom, a Version 7 port with enhancements.

Seems that real-time should be pretty easy to implement: write a device driver to handle the incoming data (assuming it's interrupt driven). The driver can turn off interrupts and buffer the data somewhere a lazy user process can get to it using read(2). You can even time-stamp the data on the way in. Actually, when you think about it, a device driver is a real-time application.

As for the fair-share scheduler, well I've worked with it, and it can't quite be used for real-time simulation. The percentage given is a goal that the scheduler attempts to maintain by balancing clock ticks whenever the scheduler is called. You still may have to wait your turn in line to get the CPU when you need it, unless you never relinquish it. If you set the CPU allotment to 100%, you will still lose the CPU when you attempt to perform I/O, and you may not regain it for a second or more. Seems like a better idea to modify the scheduler so that a superuser process can call a system routine to set a process table flag that states: This process preempts all other processes when it's ready to run. Of course, you can only have one real-time process running on the system at a time; otherwise, there'll be contention unless you've programmed everything real good.

Oh, Yes, MASSCOMP also sells a real time version of System V called RTU.

Pat Wood

os386/expert.query #24, from tanj, Wed Dec 3 20:00:26 1986. A comment to message 23.

That version of Unix is *heavily* modified, and not generally available. It has a level of tasking below processes, multi-CPU features, and reliablity features even in the opearator console process to help them diagnose trouble accurately. Last I saw it had 99.999% up-time.

387-WEITEC???

os386/expert.query #27, from jaymartin, Sat Dec 6 01:14:49 1986.

Does anybody know when the 80387 is going to be released? Is it software compatible with the 8087 and 80287? And how about the Weitec Math coprocessor that Intel keeps giving benchmarks for. Is it software compatible with the 80387? What is going to the price tag on these processors? I have seen results showing the Weitec twice as fast as the 80387. If this is true, who needs the 80387? Floating Point speed is critical to many scientific applications and the 80287 is pathetic.

os386/expert.query #28, from intel, Sat Dec 6 04:23:06 1986. A comment to message 27.

The 80387 is fully compatible with 8087 and 80287 and the IEEE P754 Rev 10 Numerics standard. New instructions include support for all Trig operations and more importantly it accepts trig arguments > 2 pi.

The 387 is sampling now and will be generally available in Q1/Q2. It is 6-8x th e performance of a 8 MHz 80287.

The Weitek chip set is not compatible with anything. It is intended to be used for Engineering Workstation. Intel has worked with Weitek in developing an interface chip and providing C and Fortran 386 compilers which generates Weitek code. The Weitek chips and the compiler (GreenHill's) are available to OEMs now.

Clif Purkiser

os386/expert.query #30, from rnelson, Sat Dec 6 12:11:26 1986. A comment to message 27.

Also, the Weitek chips are "4-bangers," i.e., add, subtract, multiply, and divide. The 387 has all the good transcendental support, sqrt, etc.

os386/expert.query #29, from billn, Sat Dec 6 11:50:43 1986. A comment to message 28.

The Weitek chips (plural) are large, expensive and *fast*. The 80387 is a big jump up from the 287 in performance and should hold the fort for all but the scientific compute hogs. If I remember right, the 387 was spec'd out at about 1.7 MFLOPS. That kind of speed should hold 90% of the market. BillN

EGA IN MULTITASKING

os386/expert.query #36, from jcockerham, Mon Jan 5 18:44:09 1987.

I have been thinking a lot about the EGA and how it might function in a multi-tasking OS. It is not a very friendly adapter because the latches have to be saved across a context switch for the screen. Does anyone out there think that saving the EGA hardware state is a reasonable proposition? Are the designers of the OSs for the 386, and the 286 for that matter, planning on a forced context switch for a virtual 86 task which owns the EGA when it might be expecting the hardware in a particular state. The plane latches are loaded on every CPU read from video memory. In order to save the latches, they must be stored into EGA memory and read out. This means that the OS must reserve a piece of the

EGA memory space (1 byte) for this operation. If the designers agree that the EGA will only change hands across an operating system call, which is in itself not too safe, then the latch stuff can be gotten away with -- not saving the state.

os386/expert.query #37, from mced, Tue Jan 6 13:23:29 1987. A comment to message 36.

Sounds like a reasonable answer --- In my discussions with Microsoft about designing new graphics boards, their first comment (shouted loudly in chorus) was "Let us read whatever we can write!". Of course, your solution assumes that each application owns the entire EGA screen when it is active - no multiple apps in windows or shared screen territory.

os386/expert.query #38, from jcockerham, Tue Jan 6 19:43:47 1987. A comment to message 37.

However in this case, let us write and read whatever we can read and write, or something like that <grin>

os386/expert.query #39, from dathomas, Wed Jan 7 01:49:30 1987. A comment to message 38.

MultiUser Graphics? How are multiple dsiplays supported? Mapped to different physical addresses? Has anyone done this?

os386/expert.query #40, from jcockerham, Wed Jan 7 18:52:09 1987. A comment to message 39.

Multiple EGAs can be supported by disabling the BIOS in one, and fixing the addresses for the page of video memory. More than likely some of the OS will perhaps, provide some form of emulation that lets the application think that it has the adapter when it really does not. This is not multi-user graphics per se, but a mechanism to allow several routines to share a resource. I am just trying to think about the right way to do this. With the EGA, as currently defined, it will not be easy to integrate into a multi-user env.

os386/expert.query #41, from mced, Wed Jan 7 21:03:28 1987. A comment to message 40.

Part of the problem is that the obvious solution doesn't work. The EGA performs some funny internal operations depending on all data written to it. Although I haven't thought about it extensively, you might get into trouble trying to virtualize an EGA by simply (assuming a Virtual 86 mode program here) mapping the EGA's address space to another physical location and then updating the REAL EGA when that task is active (or owns the display, or whatever). That sort of approach would work OK for a monochrome or CGA kind of display, but not for an EGA (I don't think).

Fortunately, I have the luxury of spending most of my time designing operating systems for graphics coprocessors and producing graphics boards which allow each application in a multitasking environment (and even the multitasking environment itself) to totally ignore the fact that they are running in such an environment. Applications just leave the driving to us.

os386/expert.query #42, from jcockerham, Thu Jan 8 18:35:19 1987. A comment to message 41.

How would one provide the 4 parallel bit planes and the soft fonts if one attempts to virtualize the EGA? Conceivably, the task which attempts to talk to the EGA could be suspended until it gets the hardware, or some such thing. But like it or not, the EGA is not going to fit into the 386 architecture very easily at all if applications expect to roll their own graphics.

NOW I CAN REMEMBER...

os386/expert.query #43, from mced, Thu Jan 8 20:19:36 1987.

... what the problem is.

If the OS simply redirects the EGA output to another RAM location and then updates the real EGA in the same manner when that task becomes active, you'll be OK until your application tries to READ back data which was (presumably) modified by the EGA in weird and wonderful ways when it was written.

Perhaps the only way to do it is to write a software emulator which can trap all EGA memory accesses and modify the results properly each time. It will probably have the undesirable side effect of making the system speaker say "oink oink" each time, however.

os386/expert.query #44, from bilin, Thu Jan 8 21:25:04 1987. A comment to message 43.

The word I have on that emulation is 1000:1 slowdown. Bit of a problem, what? BILIN

os386/expert.query #46, from intel, Fri Jan 9 02:47:12 1987. A comment to message 43.

It is correct that virtualizing the EGA is one of the toughest tasks in designing multitasking OS that does virtual EGA, I imagine Softguard can shed some light on how the deal with nasty issues like write-only registers that do strange things.

This leads me to a question are there any good books on programming/understandin g the EGA. My favorite PC book the Norton book does not get into any real details

However there is hope with solving this problem also. Intel has developed a nifty graphics chip called the 82786 which provides fast support for hardware windows. My crystal ball says that the 786 with some fancy software should be help virtualize the EGA.

os386/expert.query #47, from rduncan, Fri Jan 9 03:28:14 1987. A comment to message 46.

It would be a shame to see anyone waste the power of the 82786 on virtualizing the EGA. Some nice articles on EGA.programming are finally appearing. There have been several in PCTJ in the last few months, and the lastest issue of Programmer's Journal has a nice one too. For a general introduction see Wilton's article in the 1985 IBM PC issue of BYTE.

os386/expert.query #48, from bilin, Fri Jan 9 07:45:28 1987. A comment to message 46.

The current info available on BIX is listed in grafic.disp/specs #6 courtesy of Barry Nance (barryn). BILIN

os386/expert.query #49, from mced, Fri Jan 9 20:55:15 1987. A comment to message 48.

I'd have your crystal ball hauled in for a tuneup, Mr. Purkiser. Today, in fact is the first anniversary of the day I held an 82786 in my hot little hands and during that year I've discovered precious little in it which would help virtualize an EGA. It has a large number of other outstanding features which make it EXCELLENT for use in 80386 systems, however. The chip's basic ability to remove any hardwired association between physical display RAM and the data displayed on the screen make it ideal for multitasking environments. Each app requests its own portion of screen memory (as one would do with a malloc() call in system memory) and then request the OS or graphics system to create a hardware window in which to display that data. The actual placement, sizing, and overlapping of those windows is managed by the OS external to the application - but the application doesn't care. Gone are the days of applications being asked by the operating environment to redraw a portion of their display because another window moved and exposed it. With an 82786, it does nothing more than that - exposes it!

I get to play with an 82786 in an 80386 machine on my desk every day. I can't wait to see some people out there develop some REAL uses for the pair.

os386/expert.query #52, from tanj, Sun Jan 11 06:02:32 1987. A comment to message 49.

If I understand you aright, software "windows" are

continued

redundant. Do you have any whisper of someone integrating window priority and buffer swaps with the process swapping in the OS kernel (ideally Unix) ? I rather like the idea of swapping my window buffers to disk when my process's other memory is swapped out too. The screen's overlapping windows would give a picture of processes recently active and resident. Combine this with hardware managed virtual memory, and most of the struggle with MS Windows seems a poor investment.

VIRTUALIZING EGA/CGA VS. MULTITASKING GRAPHICS

os386/expert.query #53, from dtuttle, Sun Jan 11 15:40:56 1987.

I'll take a contrary position on this issue, just to see what other people have to say. The difficulties in multiple access to an EGA/CGA/ whatever controller is that the functional breakdown is wrong. Clean ac- cess can only be accomplished if there is a common software interface to the graphics resource, such as a windowing package, GKS kernel, or some yet-to-be-defined alternative to a direct-write-to-hardware approach.

As long as each "application" wants to use its own graphics routine, the only performance-acceptable approach is some very fancy hardware (a multiple-image controller, with built-in windowing) or more than one EGA card -- one per application, each with its own screen, or with some means for choosing which screen will be actually displayed. The multiple-card approach would require cards with a "settable" address range, so that the physical addresses would not overlap but could still be mapped into the virtual space of each task or user.

>*<- Dave T.

os386/expert.query #54, from mced, Mon Jan 12 10:05:34 1987. A comment to message 53.

> EGA virtualization

I agree that, provided we start with some rules and follow them, we're OK and that there are many acceptable sets of rules from which to choose. But the problem is supporting all the EGA-specific software that's already there!

> Hardware windowing

As I see it, a sensible system using hardware windowing can provide a set of visual priority and protection mechanisms directly parallel to those provided as processor/memory priority and protection using an 80386. A rather favorite topic of my fantasies, I admit.

os386/expert.query #56, from jshiell, Tue Jan 13 00:54:16 1987. A comment to message 53.

True a common software interface would be the "Least Worst" (ie there are no easy solutions) however this ignores the installed base of "Bad" software. A hardware solution while more expensive is also, I suspect, a more acceptable general solution.
Jon Shiell

PS: I mean the interface would be the fastest, simplest and could be used to allow virtualization of any graphic display. But ...

os386/expert.query #57, from billn, Tue Jan 13 00:57:21 1987. A comment to message 56.

Right. Nobody said it was perfect. But it would (finally!) get some programmers to write hardware independent code. And they would prosper. BillN

os386/expert.query #58, from jcockerham, Tue Jan 13 20:06:41 1987. A comment to message 57.

The intent of providing in this forum the discussion about virtualizing EGAs has to do with how those of us writing operating systems for the 386 will put virtual 8086 tasks (and I do mean plural tasks) under a common operating system (a VM if you will) and provide concurrent execution of the already written, and bad acting DOS applications. I think, however, that the EGA as it currently is architected will never easily support virtualization. One solution, but poor, would be to suspend any tasks that talk to the EGA through anything other than well-behaved BIOS calls or even higher levels.

os386/expert.query #59, from billn, Tue Jan 13 23:25:05 1987. A comment to message 58.

I would basically agree with that approach. 386 systems will rapidly replace their EGA displays as better become available, and apps will follow. Virtual EGA has a narrow (about a year IMHO) market window and is therefore not worth an excessive amount of effort.
BillN

os386/expert.query #60, from jshiell, Wed Jan 14 03:36:12 1987. A comment to message 59.

I disagree; the amount of effort is related to the number and popularity of programs that use it, and how long (if ever) it take the user base to outgrow (ie move up to "better hardware") the current EGA.
Jon Shiell

os386/expert.query #61, from jcockerham, Wed Jan 14 20:16:28 1987. A comment to message 60.

Even with better hardware, the popularity of EGA applications and the NEC multisync mean at least considering some form of emulation. There are a lot of users out there who will be expecting to run graphics and something else at the same time

os386/expert.query #62, from dtuttle, Thu Jan 15 00:13:07 1987. A comment to message 61.

The really important question is whether any significant number of people will want to run MORE THAN ONE graphics-interface application. If all you really need is one "old-style" interface plus some small number of well-behaved text-mode interfaces, it should be possible to manage without fancy (read: expensive) extra hardware.

In the slightly longer term, when a good hardware-independent interface has been defined AND ADOPTED by the majority of the software, it will still be possible to dedicate an "old-style" EGA controller with its own screen, to the important application which needs all of the interface capabilities. It's even likely that the "digital HDTV" trends will eventually give us monitor which can accept more than one video feed, with built-in "windowing".

In the meantime, the operating systems will probably have to make do with "stealing" a few-line-high window at the top or bottom of the screen, for well-behaved text messages only, while allowing a single graphics-mode task at a time. It's easy to protect the display RAM - just don't map it into the address space. This also gives the system an automatic trap on any attempted access.

>*<- Dave T.

os386/expert.query #63, from jcockerham, Thu Jan 15 21:39:53 1987. A comment to message 62.

The EGA even has that now, with its split screen. I agree that the definition will have to come, and things will be painful for a while after its inception.

EGA AND 386

os386/expert.query #67, from matt.trask, Mon Jan 19 08:03:51 1987.

We currently have the EGA fully supported in the Vpix product and it was pretty straightforward given the 386's IO permission mapping. The real question is what will the users want to be able to do with it?

os386/expert.query #68, from jcockerham, Mon Jan 19 13:06:58 1987. A comment to message 67.

That's very good. But tell me, how are you allowing two DOS sessions to run to the EGA simultaneously? That's more what this thread has been about.

os386/expert.query #76, from matt.trask, Fri Jan 23 08:18:28 1987. A comment to message 68.

I understand the focus of this thread - thus my question: "what will the users want?" Do they expect Sun windows on an EGA? Is full screen swapping sufficient? Will they be satuisfied to run a single foreground graphics task? I think we all agree that new hardware and graphics standards will remove the problems associated with current EGA hardware - I can see a 386/786 solution that will provide multiple EGA compatible sessions in windows on a large screen display.

So, being that there are no users (yet) for multitasking DOS products, how do we find out what they want while we wait for newer hardware? Vpix currently supports a single ill-behaved EGA graphics task in the foreground on the console for the beta release and we intend to support multiple EGA tasks in the final release and I believe that the user will just have to "pay the price" performance-wise in order for this to happen.

Opinions?

os386/expert.query #78, from billn, Fri Jan 23 10:49:25 1987. A comment to message 76.

Were it me making the decision, I would support a single EGA emulator session, foreground or background, displayed or not as the user wishes. This should satisfy the majority of users and give an incentive to all to upgrade software for the better display hdw. For a short while it might cause some restriction, but would minimize performance impact and the service where absolutely needed.

Consider the tradeoff. If the EGA emulation slows the screen display as little as 50:1, how many people could stand running two slow apps at one time. Given the rumored 500:1 ratios, even one would be a problem.
BillN

os386/expert.query #79, from jcockerham, Fri Jan 23 19:47:04 1987. A comment to message 78.

I think that the ratios will be even worse than 500:1, depending on how close to an EGA the emulation needs to provide. Certainly the primitive BIOS services can be written, with little if any performance penalty, because the operating system will have to provide basic screen services. The bugagoo is these very ill-behaved applications, including Windows that insist on rolling their own, by directly manipulating the hardware. I appreciate the opinions expressed by all and hope that someone smarter than I will have to create this software.

As for 82786 emulation of the EGA, that will be no piece of cake either. The 82786 does not provide the bitplanes present in the EGA. But I think any further discussion of the 82786 EGA emulation issue belongs in the graphic.disp conference.

os386/expert.query #70, from jshiell, Tue Jan 20 01:27:48 1987. A comment to message 67.

How is the performance when a background task (such as EGA PAINT does a rep movs with bit-bit operations enabled ??), or do you only support "well behaved" EGA users in the background. Jon Shiell

PS: I would guess that you must see slowdown of between 100 and 500 times for the above worst case.

386 PRM ERROR

os386/expert.query #69, from msokol, Tue Jan 20 01:24:03 1987.

Is the Intel 386 PRM correct. Is REP MOVSW really faster that REP STOSW? Marc

os386/expert.query #72, from tpennello, Wed Jan 21 01:52:35 1987. A comment to message 69.

I had also heard this was true, although as a rumor. I used the Phar Lap assembler and debugger to run a protected mode 386 program on a Compaq. It did the following
repeat 2000 times [rep (stosd | movsd) with ecx = 20000] All constants are in decimal. The times:
stosd = 13.31 seconds
movsd = 21.19 seconds time with a stopwatch.
Stopwatch on when typing "g" in the debugger; off when debugger responds that program terminated normally.

I haven't tried the test on my Intel 386/20 UNIX box but I expect similar results. All data was aligned.

I also tried the test using movsw and stosw (2-byte operations) instead of four-byte operations. The results: NEARLY THE SAME!!! (perhaps .1 second different). Moral: for aligned data, always use 4-byte opns.

os386/expert.query #73, from intel, Wed Jan 21 02:03:45 1987. A comment to message 69.

I asked the microcode boys about this sometime ago and the answers is Yes the PRM is correct Rep MOVSW is faster than REP STOSW. However, I will double check this again. Clif Purkiser

os386/expert.query #74, from jcockerham, Thu Jan 22 20:30:14 1987. A comment to message 70.

The performance will be very slow. Again, because of the EGA hardware, and IBM's decisions about that board, the EGA will be difficult to impossible to virtualize.

os386/expert.query #75, from jshiell, Thu Jan 22 22:54:37 1987. A comment to message 69.

Mark:

I ran a test on a Compaq/386 in real mode and got the following results:

Operation	Cycles per 4B moved
MOVSW DI=SI+4KB	16-17
STOSW	10-11
MOVSD DI=SI+4KB	8-9
MOVSW DI=SI+4	8-9
MOVSD DI=SI+4	4-5
STOSD	5-6
MOVSD DI{1006}, SI{1001}	19-20
STOSD DI{1006}	15-16

from the above it appears that the spec (4c/DW for Movsd, 5c/DW for Stosd) is correct. Also the difference between the +4 and +4KB cases is caused by the trashing between src and dest static col. in the compaq, the +4 case is approx. pure zero wait state and +4KB is approx a pure 2 wait state case.

Jon Shiell

os386/expert.query #80, from mced, Fri Jan 23 20:07:30 1987. A comment to message 79.

Well, I guess the emulation of a single EGA in the foreground when you have a physical EGA installed was a "given" - I assumed that when you said VP/ix did it I assumed there was something more than be able to let the hardware do what it does anyway. But I think that the ability to support multiple EGA applications is pretty vital - if for no other reason that the fact that it is pretty tough, if not impossible, for a typical user to determine by inspection whether his applications software is "clean" or not. Most folks just don't know and don't want to know.

As for taking a performance hit, that's quite certain to be a problem if you try to do something like run two copies of Windows (i.e. skip the Clock application, you'll be better off with a sundial) but I aon't think many people will do that. The trouble is that display errors are unforgiving - if ANYTHING is wrong, the user is discomfited. If my normal, nice BIOS application does ONE tiny little tweak to the EGA hardware (or at least a small number of them) I'm probably NOT going to notice the performance penalty but I WILL notice if my screen looks funny when I switch to that app.

continued

Moral: Sometimes you can do it fast, sometimes you can do it slow, but you ALWAYS have to do it right.

os386/expert.query #81, from jshiell, Fri Jan 23 21:41:33 1987. A comment to message 79.

I Agree with your comment that further discussion of using the 82786 to emulate and EGA belongs in another conf.   Jon Shiell

os386/expert.query #82, from tanj, Mon Jan 26 03:33:53 1987. A comment to message 80.

Surely the summary of this thread so far is that EGA is a tar-pit for 386 systems which aim to run DOS guests, unless they do so by suspending all but one at a time. I hope someone from companies like Interactive or Software Link has been thoughtful enough to get on the phone to AST, Hercules, Compaq (?), and suchlike, to explain the opportunity to make an upgrade of the EGA which gets us out of this mess. After all, most folks who are buying a new 386 box are buying a new adapter too.

The EGA problem exists and is thoroughly embedded in existing applications (not least because IBM never seems to understand that mediocre slow BIOS functions are of necessity bypassed). An EGA-like board which had a mega- byte of memory (ie. 4 virtual EGA spaces) and a display engine to select windows to be displayed would be the right product at the right time and make a few bucks. No-one with money to buy a 386 this year will be interested in software emulation 500 times slower, and next year the multi-EGA will be as cheap as an EGA now.

os386/expert.query #84, from billn, Mon Jan 26 13:24:54 1987. A comment to message 82.

Good suggestion. People like Chips and Technologies and ZYMOS should listen. BillN

PAGED VIRTUAL ?

os386/expert.query #83, from tanj, Mon Jan 26 03:33:54 1987.

Does any 386 OS now announced or imminent definately support paged virtual ?

os386/expert.query #85, from ksarno, Mon Jan 26 23:43:00 1987. A comment to message 83.

The INTERACTIVE Systems (my employer) port of Unix V.3 for the 386 which was done in close cooperation with folks at Intel is now in Beta test (available from Intel) in around 50 sites and fully supports demand-paged virtual memory using the AT&T 'regions' abstraction initially introduced in Unix V.2.2.

os386/expert.query #86, from jshiell, Tue Jan 27 04:55:50 1987. A comment to message 83.

In addition to the Unix V.x Ports (all of which will be supporting paging as far as I know). Softguard will be supporting paging in their VM/386 product. Note that in a PC type system paging will NOT be very important because the I/O system is very limited, thus paging will be SLOW. I would guess a rate of maybe 2 pages per second would be very noticeable.   Jon Shiell

8Ø386 OPERATING SYSTEMS

os386/expert.forum #1, from pharlap, Wed Nov 5 21:14:14

The 80386 certainly has attracted a great deal of attention from OS developers. Here is a list of announced (but not shipped) OS's for the 386:

Product Name	Company	Description
DeskView 1.3	Quarterdeck (213) 392-9701	Runs multiple MS-DOS applications each with 640K of memory. Also a windowing system. AVAILABLE NOW(!)
Merge-386	Locus (213) 452-2435	Allows DOS applications to be run under the Intel/Interactive UNIX Port. Uses virtual 8086 mode.
PC/MOS-386	Software Link (408) 998-0700	A multiuser/multitasking OS that can run both MS-DOS applications and protected mode applications. Available Feb. 23, 1987.
386|DOS-Extender	Phar Lap (617) 661-1510	Runs 32-bit protected mode applications under MS-DOS 3.X. Available Dec. 1986.
UNIX	Interactive (213) 453-8649	A port of UNIX System V to the 386. Intel is footing the bill for this one. Available 1st Quarter 1987.
VM/386	Softguard (408) 970-9240	Probably the most ambitious 386 OS project. VM/386 is patterned after the popular VM/370 mainframe OS. Runs multiple OS's as virtual machines. Will support both MS-DOS and protected mode applications. Available 2nd Quarter 1987.
VP/ix	Phoenix (617) 762-5030	Allows DOS applications to be run under the Interactive UNIX port or XENIX-386. Uses virtual 8086 mode.
XENIX-386	Microsoft (206) 882-8080	XENIX for the 386. Available 2nd Quarter 1987.

Here are some rumored OS's for the 386:

Concurrent DOS-386 - A 386 version of DRI's multitasking MS-DOS compatible operating system.

Other UNIX ports - Whenever a new chip is introduced, there seems to be at least three or four UNIX ports done for the chip by various companies. Expect the something to happen with the 386. A Berkeley 4.2 port seems like a good bet. Some of the new Sun extensions to UNIX also look very interesting.

Multitasking DOS 3.2 from Microsoft -- allows multiple PC applications to be run under MS-DOS in virtual 8086 mode. Similar in concept to DeskView 1.3.

Windows for the 386 -- runs applications in virtual 8086 mode. The big advantage over the current version of Windows is that each application gets its own 640K block of memory instead of every application fighting over the same 640K bytes of memory.

DOS 5.0, Advanced DOS, 286DOS, ADOS, etc. — By what every name you call it, this is the next version of MS-DOS from Microsoft. Supports 286 protected mode and multitasking. Will run existing 8086 PC applications but only_one at time. Since the 386 can run 286 code, Microsoft will pitch it as a 386 OS also. Use it only if you love segment registers and programming in the long memory model. Available sometime in the first half of 1987.

DOS 6.0 -- True 32-bit protected mode MS-DOS for the 386. Bill Gates was quoted as saying in ComputerWorwhat there won't be a DOS 6.0 until the end of 1987. Don't bank on it after what happened with Windows and DOS 5.0.

True Blue DOS -- Multi-tasking, protected mode, klone-killer, virtual- machine DOS from IBM that is burned into ROM and uses custom 386 instructions. Many strange rumors are flying around about this one, but it is really hard to believe any of them.
Sources: PC Week, InfoWorld, MicroBytes, and ComputerWorld.

P. NORTON AND 82786/8Ø386

os386/expert.forum #2, from fheilbronner, Mon Jan 12 14:04:06 1987.

In the 6-JAN-87 issue of PC Week PN (Peter Norton) discusses the comparitive strengths of the 82786 vs the TI 34010. PN says "if it's easy to implement virtual windows with the chip (82786), it's just as easy to implement entire virtual displays — virtual EGAs, if you will". Well I guess we will, my question is, how *easy* is easy? Most of the conversation here has been on how complex the virtualization of the EGA is. Comments?

os386/expert.forum #3, from billn, Mon Jan 12 15:38:13 1987. A comment to message 2.

Some of the discussion about the EGA virtualization has gone on in grafic.disp (unabashed plug)/processor. It seems that the 'write only' registers and the way the board handles bit planes are *really* difficult (read lots of cycles) to virtualize.

EGA can be looked at in two ways: As a product (board) requiring hardware emulation OR as a display definition (640 x 350 x 4) with 64 possible colors. Emulating the former is a bear, doing the latter ought to be reasonable.

If Peter Norton is suggesting the former (hardware) emulation is easy, he knows not whereof he speaks. Basically I'm not as suprised as some because the internals of the EGA are trickey to say the least and PN's columns have become disappointing of late.

BIlN

os386/expert.forum #4, from mced, Tue Jan 13 00:12:02 1987. A comment to message 3.

The 82786 provides NO (with a capital NOTHING) support for the virtualization of the EGA. Since one can write to an EGA display RAM location and have the EGA modify that write using its hhardwarKe (e.g. make the written value be XORed with current RAM), a sequence like

```
     MOV  ES:[BX], AL      MOV  AL, ES:[BX]
```

must, when emulated, return the properly modified value in AL. On an 80386, you'll have a hard time finding ANY graphics pr^ocessor which can "virtualize" the required hardware modification fast enough (just a few nanoseconds) to perform properly.

os386/expert.forum #5, from jcockerham, Tue Jan 13 20:00:09 1987. A comment to message 4.

However... In a virtualized EGA, the access to the RAM would be flagged and the emulating routine would somehow (And I really do not know how) set this up. One idea I had was to come up with an operating system call and maybe use the trap on the 80386 to allow execution of one instruction in the user's code and then trap back to the operating system so that the virtual EGA would be able to reset for the next read/write access to video ram. In the above example AL would be set with the correct value the emulation reset in the mov es:bx,al. The instruction would be restarted and after its execution, the emulation would trap on the next instruction the "mov al,es:bx". The correct plane would be mapped into the video ram space and the instruction would be restarted. You have to reset the emulation trap after restarting instruction so that the reads will set the emulated plane latches and perform color comparison and the cpu writes will set all of the mapped planes. This is a real bear!

os386/export.forum #6, from mced, Thu Jan 15 10:49:23 1987. A comment to message 5.

You're right about that method solving the problem... but causing a protection violation on every RAM access (and I/O access) to fake it out makes me want to cry.

IN THE BEGINNING...

users386/compat.hard #2, from rduncan, Mon Nov 3 23:56:44 1986.

I've heard rumors that Intel is on the verge of announcing a 80386 accelerator board for the PC/AT. Some say that it will be displayed at Comdex, and that it will cost several thousand $. Anyone have some authentic info on this?

users386/compat.hard #5, from inboard, Tue Nov 4 14:33:35 1986. A comment to message 2.

I think I can help. Intel has already announced the product; it's called Inboard 386/AT. It's an enhancement board for ATs and compatibles.

It uses a replacement architecture for compatibility. Depending upon the program, it delivers 2X performance increase over an 8MHz AT.

There's a 0-wait state cache and up to 3 MB of 32-bit memory (1 MB on Inboard itself and 2 MB on a piggy-back module). There's a socket for a 387 when it becomes available; until then, there's a 10MHz 287 module available. List price is $1995 with 0K. First public showing is at Comdex next week. Shipments start in January.

I'll be happy to answer any questions. My name is John Beaston and I'm the 386 Program Manager for Intel's Personal Computer Enhancement Operation.

users386/compat.hard #7, from villi, Wed Nov 5 16:23:30 1986. A comment to message 5.

This sure looks nice. I already have an Intel AboveBoard with 1.6 Megs on it in my AT, using it as a RAM disk. I presume there will be no problems using it with the InBoard. Am I correct? Also, is Intel going to be supplying the software to run PC-DOS in virtual mode on the board? If not, who is? Although I'm gonna be developing software (a compiler) for the 386, I still want to run PC-DOS on the board, RAM disk and all.

users386/compat.hard #8, from inboard, Thu Nov 6 1:02:32 1986. A comment to message 7.

You bet Inboard is compatible with AboveBoard. The Inboard memory is all extended memory and we will be supplying an expanded memory emulator. So you might want to reconfigure your AB memory for extended. That's just a suggestion, there are several ways to go; use leave the AB as expanded memory and use the Inboard memory with VDISK, etc.

I can't comment on 386 control software for Inboard. I can say that we are working with every vendor I know of who's doing such software. Watch the press for announcements....

continued

users386/compat.hard #9, from villi, Fri Nov 7 20:37:21 1986. A comment to message 8.

This is just what I wanted to hear. My AboveBoard is already configured as extended memory, since we also use Xenix/286 here. Say, this InBoard looks great. When can I get my hands on one?

users386/compat.hard #22, from jbarrett, Sat Dec 6 04:00:18 1986. A comment to message 5.

I own a Z-248 (Zenith PC/AT clone) configured as follows: 1 CDC 40Meg hard drive 1 Segate 4051 40 Meg hard drive 2 Z-445 Fast memory cards (0 wait states) 1 Intel Above Board PS/AT w/piggybacked memory 1 80287 8MHz coprocessor 1 Tecmar EGA card My question is will the Inboard 386 work in this machine without overloading the power supply and/or crashing in flames (figuratively). Is itr really worth the extra expense for the extra umph? According to Norton's SI ver 3.00 my machine all ready marks at 9.2 index. Is thier any literature on the performance running BYTE'S infamous benchmarks?

users386/compat.hard #23, from inboard, Sat Dec 6 13:11:55 1986. A comment to message 22.

There shouldn't be any problem with the power supply; every AT compatible that we've looked at had plenty of juice left. There might be a mechanical problem with your machine however. We have not tested the Zenith machines yet. Since they use a non-motherboard approach, we might have cabling problems. It will be hard to tell until we try it. Call our technical support hotline (800-538-3373) and ask if it's been done yet. [If it's not yet been tested, they probably can't give you a schedule on when it will be - there are soooo many machines and soooo little time.... Just try the hotline ocassionally.]

Regards, John.

WAIT OR PLUNGE?

users386/compat.hard #14, from curtf, Tue Nov 18 09:05:02 1986.

I just returned from COMDEX, where I saw quite a few 80386-based products. Most of these were being "announced" or "previewed," with delivery promised anywhere from Q1 '86 to Q2 '87. I also saw a *LOT* of 12Hz 80286 products, most of which are now being shipped. My question is this: Given the ready availability of 80286 machines, and the un-availability of 80386 machines (and I've not even mentioned OS and application software for the 80386), would a company which needs high-speed microcomputing be better served by buying an AT-clone now, or by waiting (6 weeks or 6 months) for an 80386-based machine. I don't really want an analysis of the situation of the purchasing company, just your reaction to the current state of the two types of machine.

Your Ob'd etc., Curt Franklin, Co-Moderator

users386/compat.hard #15, from bilin, Tue Nov 18 11:14:13 1986. A comment to message 14.

Not an easy question to answer. If they need the 386 speed then COMPAQ or Advanced Logic Research (ALR) are shipping 386 machines now. ALR actually beat Compaq announcement by one week and has been shipping since August 1. But there is a cost issue. Right now, 386 machines cost more per MIP than 286 machines. If they a buying a bunch, this is an issue.

My personal opinion is to go for a 386 machine because of the ultimate benefits of the processor capability. But for routine non-power user use, an inexpensive AT does the job, is cost effective, is available and well debugged. Don't go 386 if an AT will do the job. BilIN

users386/compat.hard #16, from inboard, Tue Nov 18 11:24:58 1986. A comment to message 14.

Forgetting my corporate allegiance, we have a lot of experience on 12MHz ATs and 386 machines. Our compatibility lab has tested several 386 machines all with excellent results with both hardware add-ins and software. Our experience with 12MHz ATs is mixed.

The design of the 12MHz AT has a direct bearing on how well it will work with add-in boards. Some designs simply crank up the clock. While these machines give the best performance (both processor and bus speed up), they're the worst for compatibility. Designs which slow down the bus are definitely the best and the performance impact is minimal.

Overall, I'd wait. Given the problems we've had with 12MHz ATs and the fact that 386 systems are shipping now (Compaq at least and others in Q1), the 386 is the right choice. Virtual-86 mode software will also be shipping in Q1 and it won't run on an AT at any speed. Also, if you're thinking about protected-mode DOS, don't forget it'll be slower than regular DOS. A 386 machine will more than give back that lost performance.

users386/compat.hard #19, from bilin, Tue Nov 18 15:11:11 1986. A comment to message 16.

I'd say that's an oversimplified answer. Good 12 and 16MHz AT's are in existence and have been tested. There is a good case for those machines that can be made if it is not for a power user. Cost/performance and availability are still real issues. In my opinion, there is still market and room for both. BilIN

users386/compat.hard #17, from skluger, Tue Nov 18 11:44:37 1986. A comment to message 14.

IMO a 16 MHz AT clone (heck, even a 12 MHz unit) will deliver a much better cost/performance ratio until at least 4Q87. That is, 80286 machines, of course. The 386 will not be a great improvement until 32-bit OSs are written and used, and until people migrate from their 8-bit 8088 applications into the 32-bit world. Given the software base and the reluctance to switch, the latter may never happen at all.

KEYBOARD PROBLEMS WITH COMPAQ 386

users386/compat.hard #18, from terjem, Tue Nov 18 15:04:30 1986.

I have just started to use a 386 beast here in Norway, and I have found the following problem: I am using Compaq's Enhanced Keyboard (100+ keys). This keyboard will not run with IBM's DOS 3.2 Keyboard Support program; KEYBNO.COM, or rather, it will run, but all the extra keys are disabled, and the special combinations which we use here to get both English and Norwegian letters dosn't work.

Terje Mathisen

users386/compat.hard #20, from villi, Mon Nov 24 20:37:30 1986. A comment to message 18.

Lest you Americans forget, the problem mentioned by terjem is a very serious one when it crops up. Manufacturers must bear in mind that a lot of people don't speak English as their native tongue.

In Iceland, we've got 10 additional characters (aside from the English 26) which must be entered on the keyboard with a key combination. If a machine can't support this, it can't be sold at all here, and I suspect that the same is case in most other European nations.

PROGRESSIVE ST/386 BENCHMARKS

users386/compat.hard #21, from mwerner, Wed Nov 26 05:47:00

Well here go's a very unscientific test, of the rough performance figures for the Progressive Electronics ST/386 AT compatible. First a short description of the system. The Progressive ST/386 that we have here at our firm are from Computer Classifieds in Miami Florida. The ST/386 comes in an XT form factor mother board with 3 eight-bit slots and 3 16-bit slots for PC compatible cards. There are two unique slots on the right hand side of the card, which accomodate the 80386 cpu card and the 640k ram card. The board is designed to allow the user to plug in either a 12 Mhz 80286 or 14.8 Mhz 80386 depending on your needs. The RAM card is composed of 640k of 120 ns RAM, and I am told that a 4 megabyte card will soon be ready. A 16 megabyte card and Very High performance EGA card are also in the works. The system that I am testing has no math coprocessor at the moment, but I will add one very soon. While on the subject of math chips, the cpu card on the 80386 has a socket for one and jumpers that allow the user to select several different clock speeds at which to run the coprocessor. One very nice feature of the ST/386 is that all of the setup procedures are contained in the BIOS ROM and if no setup is stored in the battery backed up RAM, the system presents the operator with a setup menu that covers all of the varoious stuff you need to setup such as video card type, time/date, floppy drive types, hard drive type, speed of cpu and number of wait states, and smooth scroll and screen blanking time. There's a lot more stuff to mention but this is really not a review! Now for some benchmark times. I used several common programs that exist in the public / and are easily obtained from most RBBS systems. If anyone has something better, and would like me to run it I would be glad to do so, just send a disk to my resume address and I'll run it and get back to you. The Times are as follows;

```
#1      Norton's SI shows 18.0 to 18.7
#2      CPU2 By S.Davis and K.Levitt  Mixed Test 1.21
sec.
        Clock = 39.42
        Sieve of Eratosthenes 10 reps 00.44 seconds
#3      P.C. Magazine Prime Number Test  50 primes
00:00:06                 seconds
#4      Looping test 50,000 Loops 2.5 seconds
#5      Long Integer Factoring Program  1394761 at 871

iterations 2.8 seconds
#6      Fibonacci Number Generator  10 iterations
        Fibonacci(24)=46368 7 seconds
#7      MicroDesigns Benchmark Program reports a 729%
                                          increase in
performance over a standard PC.
```

Well that's it for now, if you got any more questions either bix mail them or call me at the office. MWerner P.S. The text editor on the bix system is real crap! so for the real name of the vendor of this product read the following. Computer Classifieds Inc. They are in Byte This Month.

SPEAKING OF RF

users386/compat.hard #30, from mced, Wed Dec 31 00:12:15 1986.

I would suspect that a lot of the problems people are seeing with 80386s are similar to those I'm more familiar with in graphics products - in the case of the TI 34010, I'm dealing with a CPU running at 50 MHz! Especially in RAM interfaces, the Intel chips (80386 and 82786) run fast, sharp clock signals containing edges with lots of really high-frequency harmonic components. Nasty, noisy stuff if you're not careful.

users386/compat.hard #31, from skluger, Wed Dec 31 11:44:13 1986. A comment to message 30.

Our new graphics board was originally designed with up to 100 MHz clocks. When the prototype stopped smoking and the oscillator worked, I tuned it to 72 MHz, then walked down the street a hundred yards with my 2 meter HT and it still had a strong signal at 144 MHz (and lotsa noise +-250 kHz. yes, them beasties are noisy! monitor...

80386 ACCELERATOR BOARDS

users386/compat.hard #35, from rduncan, Thu Jan 8 03:38:43 1987.

Talked to a lady at Intel today who said the ship date for the Inboard 386 has slipped to late February. Kinda strange since working and very 'production-looking' boards were demonstrated at Comdex Nov 86. Whats the holdup, Intel? In contrast, a spokesman for Orchid Technology said today that shipments of their Jet 386 board will begin on January 28th and that a limited number of boards will be available directly from Orchid at an introductory price of $1199. The Intel & Orchid boards both have 64 KB cache memory but the Intel board can accept up to 3 MB additional fast RAM. The Intel board requires that you remove the 80286 from your AT or clone altogether, while the Orchid board leaves the 80286 also in the system and you can switch between them if necessary for software compatibility reasons.

users386/compat.hard #36, from inboard, Thu Jan 8 17:07:41 1987. A comment to message 35.

Well, it took a bit longer than expected to polish off the last of the compatibility issues in AT clones. (Some of those systems don't use the best design practices.) Don't worry though, it'll be solid when it does ship next month. [Those boards you saw running at Comdex where all in IBMs - not clones.]

The biggest obvious difference between the Orchid board and ours is the extra memory. With Jet 386, except for the cache, all memory is either on the motherboard or on expansion boards. With Inboard, the extra memory is 32-bit. It's not just limited to being extended memory; you can configure it to supply from 256KB to 640KB.

In the 1MB configuration of Inboard, you can split the memory into 256-640K conventional and 1-1.5 (or 1.5-2)MB extended memory. For both Jet and Inboard, there's a big penalty for going to the motherboard memory. It's why we give the user the option of making as much of the Inboard memory as useable as possible.

I'm not sure but I don't think Jet 386 will work in anything other than an IBM. Unless they've changed their cable scheme since Comdex, it precludes anything other than a PGA-style 286 and only IBM uses those. Inboard's cabling supports all types of 286.

COMPAQ ROLLS OUT 386

microbytes/items #453, from microbytes, Tue Sep 9 16:57:22

Compaq Computer Corp. (Houston) today officially rolled out its 32-bit machine, the Deskpro 386. Based on Intel's 80386 microprocessor, the system runs at 16 MHz but can also use add- ons and peripherals designed for 8-MHz 80286-based computers without modifications. At a press conference at the Palladium in New York City, attended by executives from software houses supporting the new machine, Compaq CEO Rod Canion said shipments to US and Canadian dealers have already begun. Canion said Microsoft will have XENIX System V/386 ready for the new machine in the first half of 1987. Bill Gates, Microsoft chairman, said a software development toolkit for XENIX is ready now. The Deskpro 386 is packaged in two versions. The Model 40 comes with 1 megabyte of RAM, 1.2-megabyte disk drive, 40- megabyte fixed disk drive, parallel and serial interfaces, three available 8-/16-bit and three 8-bit slots, and keyboard. Suggested retail price is $6499. The Model 130 differs by having a 130-megabyte fixed disk drive and one less 8-/16-bit slot. Suggested retail is $8799.

COMPAQ CHIEF TALKS

microbytes/items #456, from microbytes, Wed Sep 10 16:04:43

At yesterday's debut of Compaq's Deskpro 386, Rod Canion, Compaq president and CEO, claimed the new machine would perform most tasks two to three times faster than an IBM PC AT; if the computer were running 32-bit software, it could perform at ten times the speed of the AT, he said.

Canion said Compaq chose to use the standard 16-bit bus over a new 32-bit bus because the 32-bitter was really required only for memory. The Deskpro supports

continued

up to 10 megabytes of memory on cards attached to the system via a proprietary 32-bit connector. Canion said the performance of disk drive adapters and display adapters is limited not by their bus connectors but by associated drive characteristics or on-board dedicated processors. These cards would gain no benefit from a 32-bit connector.

When asked about the option of installing 80386 coprocessor boards in existing computers, Canion said the company had studied this idea and dismissed it as not being a very good approach. An 8-MHz 286 computer with a 386 board installed, he said, would still have pretty much the same performance as a 286 machine. Asked if a portable 386 machine might soon be available, Canion would only say, "That's an interesting question." Queried about introducing a 386 computer before IBM had set a standard, Canion said that a need existed for the computer and that the 386 represented a critical value right now. He recognized that some buyers may wait to see what extra features IBM incorporates into its 386-based system but said such features would probably be implemented on industry-standard 16-bit-bus expansion cards and would be compatible with all industry- standard machines.

Canion predicted that in six months the 80386 will be established so well that IBM will have a problem getting users to accept a new proprietary system.
--- Rich Malloy

PRODUCT PREVIEW:

microbytes/features #4, from microbytes, Tue Sep 9 17:17:24

The Compaq Deskpro 386

A high-performance PC AT-compatible system based on Intel's 80386

by Dennis Allen & Tom Thompson

About a year ago, Intel began selling samples of its latest- generation microprocessor, the 80386 (see the November 1985 BYTE, page 9). After much anticipation this processor has finally made its way into the design of several new microcomputer systems. Compaq, the Houston-based manufacturer widely known for its IBM-compatible computers, has introduced one of the first such systems, the Compaq Deskpro 386. The new Compaq machine was designed to be compatible with 80286-based systems, such as the IBM PC AT, yet take advantage of the 80386's processing power for better performance. Like the PC AT, the Deskpro 386 was also designed to run much of the existing software written for the older 8086/8088 Intel microprocessors.

System Description

From the outside, the Deskpro 386 is spartan in design. The system is housed in an IBM PC AT-style box with indicator lights, a security key, and space for up to four half-height disk drives or other storage devices. The back panel of the system unit has a 9-pin serial port and a 25-pin parallel printer port. The system comes with your choice of a standard 84-key PC keyboard or the Compaq Enhanced Keyboard, an IBM RT PC-style 101-key keyboard.

The standard configuration, called the Model 40, sells for $6499 and includes 1 megabyte of RAM, a 1.2-megabyte floppy disk drive, and a 40-megabyte hard disk. Compaq also offers a system configured with a 130-megabyte hard disk (instead of the 40- megabyte hard disk) called the Model 130, which sells for $8799.

Because no production machines were available at press time, we examined a preproduction Model 40 that had an additional megabyte of RAM (for a total of 2 megabytes), 360K floppy disk drive, 40-megabyte tape cartridge unit, and color graphics adapter. The system used MS-DOS 3.1.

Inside the machine is a 192-watt power supply, a fan, and a single motherboard. The motherboard contains a real-time clock with battery backup, seven expansion slots, the CPU, and a 32- bit slot occupied by the System Memory Board. Four of the expansion slots are full-size 8-/16-bit slots, and three are 8- bit slots, two of which are full size and one half size. Compaq's multipurpose disk controller, which is included with the base system, occupies one of the full-size 8-/16-bit expansion slots. The disk controller supports two 1.2-megabyte floppy disk drives, a 40-megabyte hard disk, and either a second 40-megabyte hard disk or a 40-megabyte tape backup unit. The Model 130 requires an additional drive controller in one of the full-size 8-/16-bit expansion slots for the 130-megabyte hard disk.

Unleashing the 80386

The CPU is a version B1 80386 microprocessor running at 16 MHz. The 80386 has built-in memory management and supports a numeric coprocessor, but the motherboard has a socket for only a 4- or 8- MHz 80287 -- not an 80387. The 80386 uses two separate 32-bit buses for addressing and data. The processor can dynamically size its data bus to handle 32-bit or 16-bit data bus operations. Also, the address bus can be pipelined: that is, the processor can perform address decoding for the next bus operation during the previous bus cycle, allowing for overlap of bus activity.

To tap the performance potential of the 80386, Compaq designed a high bandwidth CPU bus and memory bus. The CPU bus is a 32-bit non-multiplexed address and data bus. This bus provides signals for interfacing to both the 32-bit memory bus and the 8-/16-bit expansion bus. In the event of bus contention between the memory bus and the expansion bus, the memory bus has priority. The expansion bus is electrically compatible with existing plug-in cards for the PC AT. However, we did not test any plug-in cards in the system.

The memory bus provides the bandwidth necessary to take advantage of the 80386's speed and bus pipelining. It uses a paged memory architecture to improve access times. The memory bus does not include I/O status or control signals, and it is not intended to be used as a general-purpose bus. The maximum physical memory this bus can address is 16 megabytes. However, using Compaq options, you can expand the system only to 10 megabytes of RAM on the 32-bit bus.

Faster Memory

Naturally, a faster memory bus requires faster memory. For this, the System Memory Board is equipped with 36 256K-bit static- column RAM chips soldered directly to the board, for a total of 1 megabyte of memory with 4 bits for parity. Using this arrangement with 100-nanosecond RAM reduces the number of wait states required for memory access in the paged mode to nearly 0. Memory cells within the same physical page can be rapidly accessed by keeping the row address of the RAM constant while modifying the column address. For such consecutive memory fetches within a page, access times can be as low 50 nanoseconds. During nonpaged operations, access times are about 100 nanoseconds. A PC AT, on the other hand, is equipped with 150-ns RAM.

The System Memory Board has sockets for another megabyte of RAM chips, which cost $549. Additional memory must be added in 1- megabyte increments. When upgrading memory, you must change a set of 9-pin jumpers on the memory board. You can also set the jumpers to reduce the 640K-byte base memory of the system to 512K or 256K.

A special expansion board can be piggybacked on the System Memory Board to bring the total memory to 4 megabytes using 256K- bit chips. Compaq also offers a piggyback board with 4 megabytes of RAM using 1-megabit chips for $2999. This board has sockets for another 4 megabytes of RAM ($2699), again using the 1-megabit chips. A fully populated System Memory Board (2 megabytes) and expansion board using 1-megabit chips (8 megabytes) give you a total of 10 megabytes of 32-bit high-speed memory. You could also use two 16-bit boards configured with 2 megabytes each to bring the Deskpro 386 to a maximum of 14 megabytes using Compaq options. In doing so, however, you would lose the speed advantage of the 32-bit memory bus.

The Virtual Machine

An important feature of the 80386 CPU is its virtual mode. This mode, combined with memory paging, allows a real mode environment (64K-byte segments, 1 megabyte of physical address space, no memory protection) to be emulated anywhere within the 80386's physical address space of 4 gigabytes. The virtual mode also features I/O protection so that the host operating system can imitate various I/O ports. Compaq claims to have successfully "virtualized" an 8086 machine in the Deskpro 386. In other words, MS-DOS programs should run on the Deskpro 386 with little or no modification. More importantly, ill-behaved programs -- that is, programs that read or write directly to hardware I/O ports rather than using DOS functions -- should operate properly.

To access memory beyond the 640K of base memory under MS-DOS control, the Deskpro 386 uses a proprietary software driver called the Compaq Extended Memory Manager (CEMM). The CEMM takes advantage of the 80386's memory paging features to emulate the Lotus-Intel-Microsoft (LIM) expanded memory specifications in the Deskpro 386's 32-bit memory system. In effect, it virtualizes an Intel AboveBoard. You can install the CEMM and define the memory size (up to the 8-megabyte LIM limit) using the MS-DOS configuration file, CONFIG.SYS. Using the CEMM with the Deskpro 386's 32-bit memory should result in favorable speeds compared to using the LIM specifications with a 16-bit memory board.

This virtual machine arrangement promises to resolve possible software compatibility problems with existing 8086/8088 and 80286 real mode programs, at least in the single-user mode. In fact, the 80386's virtual mode will allow copies of different operating systems to execute real mode applications concurrently with memory protection and privilege control. But, for now at least, Compaq does not support host software that allows different operating systems to run concurrently.

System Speed Control

Another obstacle to software compatibility are programs using time-dependent code that relies on the computer system to be operating at a particular speed. Copy-protection schemes and certain program displays (typically games) fall in this category. Compaq's answer to this problem is the Deskpro 386's simulated System Speed Control.

The speed control is accomplished by lengthening the refresh cycles on the system bus, effectively slowing the CPU. However, lengthening of the refresh cycles is done in a way that does not interfere with direct memory access transfers or the bus bandwidth. The Deskpro 386 normally operates in an automatic mode where the CPU speed is reduced to 8 MHz -- essentially mimicking a PC AT -- each time a program accesses a floppy disk drive. The system resumes its high-speed operation as soon as the disk I/O is finished. Performance is not degraded, since the system must wait on the slower disk drive.

An MS-DOS command, MODE, allows you to manually select a system speed. You can select 4-MHz 8088, 6- or 8-MHz 80286, or 16-MHz 80386 system speeds using this command. The speed remains the same (even through a keyboard reboot) until you alter the setting, or a power-on reset occurs.

Fast Disk Drives

To complement the Deskpro 386's data processing performance, Compaq selected high-speed disk drives for the system. The 40- megabyte hard disk has an average access time of under 30 milliseconds, and the 130-megabyte hard disk's average access time is under 25 milliseconds. In contrast, the PC AT's 20- megabyte hard disk has an average access time of 40 milliseconds. Data transfer rates are 5 megabits per second (same as the PC AT's 20-megabyte hard disk), and 10 megabits per second, respectively.

For hard disk backups, the 40-megabyte tape drive has a transfer rate of 500 kilobits per second, which is about twice the speed of the drive previously offered for the Deskpro line of computers. The tape drive uses a new DC2000 tape cartridge, unlike its predecessors, which used the DC1000. However, the Deskpro 386 can read, but not write to, the older tape cartridges.

Display Adapters

The system we examined was equipped with Compaq's new Enhanced Color Graphics Board ($599), which also made use of the system's virtual mode. The graphics board provides 640 by 350 resolution with 16 simultaneous colors, and it is also compatible with IBM's EGA. Although the graphics board has only an 8-bit data path, the system cleverly relocates the board's ROM to the 32-bit RAM area. As a result, Compaq claims, graphics execution speed is increased by about four times. (The system also relocates the contents of its 16-bit ROMs to the 32-bit RAM area for speed improvement.) To go with the color board, Compaq offers a 13-inch RGB color monitor for $799.

In a departure from previous Compaq systems, the Deskpro 386 does not include a monochrome display controller. Instead, the company sells its Video Display Controller Board separately for $199. It provides the same video control as that found in other Compaq systems and is compatible with IBM's Color Graphics Adapter. The controller board can be used with either an RGB monitor (such as Compaq's), a composite color monitor, or Compaq's Dual-Mode Monitor, a monochrome monitor that sells for $255.

Compatibility and Performance

The 80386 CPU is object-compatible with 8086/8088 and 80286 code. To examine how well Compaq implemented this capability, we first ran several programs that we considered thorough in their use of memory and I/O operations. The BASICA on the machine accepted and ran the IBM PC tokenized versions of two BYTE benchmark programs (SIEVE and CALC) without problems. The programs conveniently provided us with a performance estimate.

The results of these preliminary benchmarks are impressive when compared to a 6-MHz PC AT. Generally, the Deskpro 386 ran about three to four times faster. We also compared the Deskpro 386's times to those of a PC AT specially equipped with 100-ns memory running at 11.5 MHz, and the Deskpro 386 was about twice as fast.

Next, we compiled several small C programs with Manx's Aztec C, version 3.20C, using the small memory model. We used the two floppy disk drives to compile and link the programs without any problems. Not only did these programs run flawlessly; they ran quicker than we had ever seen before.

We then ran two programs that are considered ill-behaved in their use of DOS: the XyWrite editor, version 3.05, with SideKick, version 1.52A, resident. The XyWrite editor responded correctly to the cursor and function keys, and SideKick responded properly when invoked.

Admittedly, these tests were less than comprehensive. But they do indicate a high level of software compatibility. Unfortunately, the only operating system offered for the Deskpro 386 at press time was MS-DOS. Only a true 32-bit operating system could push the machine to its limits. Compaq did say that it would offer Microsoft's XENIX System V/386 during the first quarter of 1987. [And Microsoft said a software-development toolkit for XENIX 386 is ready now.] According to Compaq, the new XENIX will be demand paged and allow multitasking operations. We did not, however, see even a preliminary version of the package.

For a Select Few

A number of folks might benefit from using the Deskpro 386. First, there are those who need the raw processing power to run very large spreadsheets or simulations. The linear address space provided by the 80386 combined with the Deskpro 386's processing speeds not only make such work possible but also bearable. And large

continued

complicated programs, such as expert systems, should run with respectable performance on this machine. There are also software developers who need a high-performance machine to shorten their development cycle. Here, the fast storage devices are particularly helpful. Moreover, the system's 80386 CPU allows developers to begin writing the next generation of software. And for others, the large storage capacity of the Model 130 and its claimed compatibility with networking software should make it a high-powered file server.

But like any new system, the Deskpro 386 is not without some disappointments. Although CAD and desktop publishing are likely candidates for development on the machine, with no I/O signals on the memory bus and the CPU's 32-bit bandwidth to peripherals effectively halved by the expansion bus, we don't see the Deskpro 386 as a serious threat in the high-speed graphics workstation arena. Also, the Deskpro 386 seems like overkill in single-user mode. Certainly, a multitasking 32-bit operating system would put the system to fuller use.

Surely, more powerful 32-bit peripherals and operating systems are coming for the 80386-based systems. For now, the Deskpro 386 appears to be a well-engineered bridge to a new generation of those machines.

[Dennis Allen and Tom Thompson are technical editors at BYTE.]

FLOATING POINT PERFORMANCE

compaq/c386 #7, from ddm, Sat Nov 1 13:23:33 1986.

I was wondering if anyone had any data concerning floating point calculations on the Compaq 386? I am somewhat skeptical that there will be a significant performance improvement (over an AT) for floating point bound programs.

After all the machine contains the very same Numeric Coprocessor that is used in the IBM AT. In particular I want to run some 3D graphics programs I've written. In a couple of cases (ie. Viewing Transformations and Clipping) I've written some assembly code that really pounds away on the numeric coprocessor. I wonder if I can expect any performance improvement on a Compaq 386.

---

compaq/c386 #8, from cdanderson, Sat Nov 1 17:52:34 1986. A comment to message 7.

>>Same coprocessor

True, but Compaq does offer an option which, like some AT aftermarket products, speeds up the 287 to a significantly faster speed than the stock 4Mhz. I haven't seen this option tested, however.

C386

compaq/c386 #15, from jwvincent, Fri Dec 5 07:03:22 1986.

Can someone here, preferably someone in the know from Compaq, help me with a few questions? I have a manufacturing program which sells for $15k+ and runs on 68020's (Apollo, HP, Sun) and superminis (VAX, DG MV, and Prime) which I would like to port to the 386. The program is fairly large (3meg object) and written entirely in FORTRAN 77. I need help with the following: Do you have a true 32 bit UNIX implemented so I don't have to segment this program but can just use the vertual memory capability or is my option to just buy enough real memory to allow it to all be resident?

Is there a F77 compiler avail that will utilize the 32 bit capabilities of the c386?

How does one go about becoming an official developer for your systems?

What is your VAR program like? My intent would be to begin to sell my software and the c386 as a bundled "turnkey" package rather than software only as currently. If you know the answers to these questions, please respond. If the response isn't for all eyes, send mail. Thanks.

---

compaq/c386 #20, from schin, Thu Dec 18 10:22:16 1986. A comment to message 15.

I belive that Microsoft was making available the XENIX 386 developers kit before the release of XENIX 386. There is a Fortran 77 compiler available for XENIX 286 but I don't know about XENIX 386.

ADDING MEMORY

compaq/c386 #23, from schin, Thu Jan 1 11:31:38 1987.

Does anyone know a CHEAP source for 386 static column 256K rams? or the specs for their 1 Mbit RAMS. I have seen 1 MBit Dynamic RAMS (100ns) advertised in BYTE for about $36 but I don't know if they are compatible.

---

compaq/c386 #24, from cdanderson, Sat Jan 3 21:12:00 1987. A comment to message 23.

I would think that dynamic and static ram would not be compatible, which seems confirmed by the fuss that everybody makes over the memory design.

---

compaq/c386 #25, from schin, Sat Jan 3 21:44:21 1987. A comment to message 24.

The difference between dynamic RAM and static RAM is that dynamic RAM requires to be refreshed constantly otherwise the values in the RAM will decay depending on the room temperature. This is accomplished by a timer in the PC which sets off a DMA channel every couple of u seconds to rewrite the contents back into ram. static ram doesn't require this refreshing so it requires much less power and may be faster. Evidently static column RAM is some hybrid between static ram and dynamic ram. Does anyone know what exactly it is? Can dynamic rams be substituted?

---

compaq/c386 #26, from tanj, Sun Jan 4 04:32:44 1987. A comment to message 25.

Static column ram is like an extended version of "nibble mode". A static memory cell is associated with each sense amplifier, one per row, so there is a whole column of static cells. So long as you do not change column address you may do a RAS-only cycle which reads from the static cells rather fast. Also called "ripplemode" by Intel, see their 51C256 data sheet. I presume Compaq just have a little circuit that recognises when a memory access hits the same column as used in the previous cycle and delivers a RAS-only no-wait access. It won't work so well in a multi-user system where peripheral accesses mix in with the program, and even on a single user machine it just helps a bit with the greedy 80386 code pre- fetch, but what the heck it is fairly cheap.

---

compaq/c386 #27, from fheilbronner, Mon Jan 5 17:19:56 1987. A comment to message 26.

Are you saying that if I use nothing but static-col RAM in a multi-user mode (Xenix for instance) that I'm not really gaining anything? Could you clarify what you said about the 386 prefetch. Specifically, under what kind of activity would the prefetch be considered "greedy"? Thanks for the info!

Jim Pauley = NOT(fheilbronner)

---

compaq/c386 #29, from fheilbronner, Wed Jan 7 13:33:55 1987. A comment to message 28.

Thanks for info Bennett! It'll take me a while to digest it but I'm sure appreciative of the time and thought that went in to your reply. Once again, I am amazed at the expertise I find here on BIX, and the real effort that gets put into helping those of us with less experience. Thanks agin' Tanj! Drinks are on me! Jim Pauley = NOT(fheilbronner)

DISKS AND DOWNLOADS

Ordering Disks of BYTE Listings

Listings that accompany BYTE articles are available in a variety of disk formats and on Cauzin Softstrip. Each disk package (which sometimes consists of more than one disk) contains an entire month's listings. If you want to order a disk package from a previous month, please call (603) 924-9281 to find out how many disks it includes. To order listings (for noncommercial use only), fill out this form and send a check or money order in the correct amount to:

BYTE Listings
One Phoenix Mill Lane
Peterborough, NH 03458

All prices include postage. Program listings can also be downloaded via BYTEnet Listings at (617) 861-6764.

BYTE issue: ______________________

COMMON 5¼-INCH FORMATS

All cost $8.95, $10.95 outside U.S.A. Annual subscription is $69.95, $89.95 outside U.S.A.

□ Apple II
□ IBM PC
□ Kaypro 2 CP/M
□ MS-DOS 8 Sector
□ Texas Instruments Professional
□ TRS-80 Model 4

COMMON 3½-INCH FORMATS

All cost $9.95, $11.95 outside U.S.A. Annual subscription is $79.95, $99.95 outside U.S.A.

□ Apple Macintosh
□ Atari 520ST
□ Amiga
□ Hewlett-Packard 150

CP/M STANDARD 8-INCH FORMAT

All cost $9.95, $11.95 outside U.S.A. Annual subscription is $79.95, $99.95 outside U.S.A.

SEND TO:

Name ______________________________
Street ______________________________
City ______________ State or Province ________
Postal Code __________ Country ____________
Check or money order enclosed for $____________

Bulletin Boards in Canada

Listed below are some computer bulletin boards that carry program listings from BYTE. Programs are for noncommercial use in connection with BYTE articles only. Some BBSs may charge an annual maintenance fee, and you must pay your own telephone charges.

Western Canadian Distribution Center (3420 48th St., Edmonton, Alberta T6L 3R5) will be supplying listings to its member bulletin board systems.

Edmonton, Alberta, (403) 454-6093

Meadowlark, Alberta, (403) 435-6579

Montreal, Quebec, PComm Systems, (514) 989-9450

Prince George, British Columbia, (604) 562-9519

Regina, Saskatchewan, (306) 586-5585

Canadian Remote Systems, Toronto

Toronto, Ontario, Epson Club of Toronto (EPCOT), (416) 635-9600

Winnipeg, Manitoba, (204) 452-5529

In addition, arrangements for BYTEnet Listings have been made with one or more system operators in the following nations: Australia, Austria, Brazil, Denmark, France, Hong Kong, Indonesia, Italy, Japan, Malaysia, The Netherlands, Nigeria, Norway, Saudi Arabia, Singapore, Sweden, Switzerland, United Kingdom, and West Germany. Contact us at (603) 924-9281 for an up-to-date list. ■

EDITORIAL CALENDAR

1987

MAY — DESKTOP PUBLISHING: An exploration of the hardware and software needed for desktop publishing, from page description languages to high-resolution printers and typesetting back ends.

JUNE — COMPUTER-AIDED DESIGN: The anatomy of computer-aided design/drafting software, the graphics display devices needed for CAD, and the data structures used by CAD programs to export data to other applications.

JULY — LOCAL AREA NETWORKS: The technology of linking personal computers together to share data files, programs, and peripheral devices.

AUGUST — PROLOG: A look at logic programming with articles on tips and techniques and explorations of the tasks Prolog is best suited for.

SEPTEMBER — PRINTER TECHNOLOGIES: An examination of the state of the art in printer technologies, including laser, liquid-crystal shutter, and ink-jet technologies.

OCTOBER — HEURISTIC ALGORITHMS: Artificial intelligence techniques for giving computers the ability to learn from experience.

NOVEMBER — HIGH-PERFORMANCE WORKSTATIONS: A tour of the technology underlying the workstations used by scientists and engineers in computer-aided engineering/design.

DECEMBER — NATURAL LANGUAGE PROCESSING: The technology of getting computers to understand the natural language of man.

1988

JANUARY — MANAGING MEGABYTES: Looking at the ways computers store and retrieve data in situations where disk space is measured in gigabytes and memory is measured in megabytes. Also a look at the new applications that mega-memory and storage will permit.

FEBRUARY — LISP: A BYTE reexamination of the original language of artificial intelligence research.

MARCH — FLOATING-POINT PROCESSORS: A look at the processors that speed the computation of mathematical operations in personal computers, including coprocessors and array processors.

APRIL — MEMORY MANAGEMENT: The hardware and software issues in managing a personal computer's memory space.

MAY — CPU ARCHITECTURES: An exploration of the latest 32-bit microprocessors, including digital signal processors and programmable graphics processors.

Announcing BYTE's New Subscriber Benefits Program

Your BYTE subscription brings you a complete diet of the latest in microcomputer technology every 30 days. The kind of broad-based objective coverage you read in every issue. *In addition*, your subscription carries a wealth of other benefits. Check the check list:

DISCOUNTS

☑ 13 issues instead of 12 if you send payment with subscription order.

☑ One-year subscription at $21 (50% off cover price).

☑ Two-year subscription at $38.

☑ Three-year subscription at $55.

☑ One-year GROUP subscription for ten or more at $17.50 each. (Call or write for details.)

SERVICES

☑ *BIX:* BYTE's Information Exchange puts you on-line 24 hours a day with your peers via computer conferencing and electronic mail. All you need to sign up is a microcomputer, a modem, and telecomm software.

☑ *Reader Service:* For information on products advertised in BYTE, circle the numbers on the Reader Service card enclosed in each issue that correspond to the numbers for the advertisers you select. Drop it in the mail and we'll get your inquiries to the advertisers.

☑ *TIPS:* BYTE's Telephone Inquiry System is available to subscribers who need *fast response*. After obtaining your Subscriber I.D. Card, dial TIPS and enter your inquiries. You'll save as much as ten days over the response to Reader Service cards.

☑ *Disks and Downloads:* Listings of programs that accompany BYTE articles are now available free on the BYTEnet bulletin board, and on disk or in quarterly printed supplements.

☑ *Microform:* BYTE is available in microform from University Microfilm International in the U.S. and Europe.

☑ *BYTE's BOMB:* BYTE's Ongoing Monitor Box is your direct line to the editor's desk. Each month, you can rate the articles via the Reader Service card. Your feedback helps us keep up to date on your information needs.

☑ *Customer Service:* If you have a problem with, or a question about, your subscription, you may phone us during regular business hours (Eastern time) at our toll-free number: 800-258-5485. You can also use Customer Service to obtain back issues and editorial indexes.

BONUSES

☑ *Annual Separate Issues:* In addition to BYTE's 12 monthly issues, subscribers also receive our annual IBM PC issue free of charge, as well as any other annual issues BYTE may produce.

☑ *BYTE Deck:* Subscribers receive five BYTE postcard deck mailings each year—a direct response system for you to obtain information on advertised products through return mail.

To be on the leading edge of microcomputer technology *and* receive all the aforementioned benefits, make a career decision today. Call toll-free weekdays, 8:30am to 4:30pm Eastern time: 800-258-5485.

And . . . welcome to BYTE country!

BYTE
THE SMALL SYSTEMS JOURNAL

McGraw-Hill

NEW

Turbo C®

Turbo C: The fastest, most efficient and easy-to-use C compiler at any price

Compilation speed is more than 7000 lines a minute, which makes anything less than Turbo C an exercise in slow motion. Expect what only Borland delivers: Quality, Speed, Power and Price.

Turbo C: The C compiler for amateurs and professionals

If you're just beginning and you've "kinda wanted to learn C," now's your chance to do it the easy way. Like Turbo Pascal, Turbo C's got everything to get you going.

If you're already programming in C, switching to Turbo C will considerably increase your productivity and help make your programs both smaller and faster. Actually, writing in Turbo C is a highly productive and effective method—and we speak from experience. Eureka: The Solver™ and our new generation of software have been developed using Turbo C.

Turbo C: a complete interactive development environment

Free MicroCalc spreadsheet with source code

Like Turbo Pascal® and Turbo Prolog,™ Turbo C comes with an interactive editor that will show you syntax errors right in your source code. Developing, debugging, and running a Turbo C program is a snap.

Turbo C: The C compiler everybody's been waiting for. Everybody but the competition

Borland's "Quality, Speed, Power and Price" commitment isn't idle corporate chatter. The $99.95 price tag on Turbo C isn't a "typo," it's real. So if you'd like to learn C in a hurry, pick up the phone. If you're already using C, switch to Turbo C and see the difference for yourself.

System requirements

IBM PC, XT, AT or true compatibles. PC-DOS (MS-DOS) 2.0 or later. One floppy drive. 320K.

**Introductory price—good through July 1, 1987*

Technical Specifications

- ☑ *Compiler: One-pass compiler generating linkable object modules and inline assembler. Included is Borland's high performance "Turbo Linker." The object module is compatible with the PC-DOS linker. Supports tiny, small, compact, medium, large, and huge memory model libraries. Can mix models with near and far pointers. Includes floating point emulator (utilizes 8087/80287 if installed).*
- ☑ *Interactive Editor: The system includes a powerful, interactive full-screen text editor. If the compiler detects an error, the editor automatically positions the cursor appropriately in the source code.*
- ☑ *Development Environment: A powerful "Make" is included so that managing Turbo C program development is highly efficient. Also includes pull-down menus and windows.*
- ☑ *Links with relocatable object modules created using Borland's Turbo Prolog into a single program.*
- ☑ *ANSI C compatible.*
- ☑ *Start-up routine source code included.*
- ☑ *Both command line and integrated environment versions included.*

Turbo C and Turbo Pascal are registered trademarks and Turbo Prolog and Eureka: The Solver are trademarks of Borland International, Inc. Microsoft C and MS-DOS are registered trademarks of Microsoft Corp. IBM, XT, and AT are registered trademarks of International Business Machines Corp.

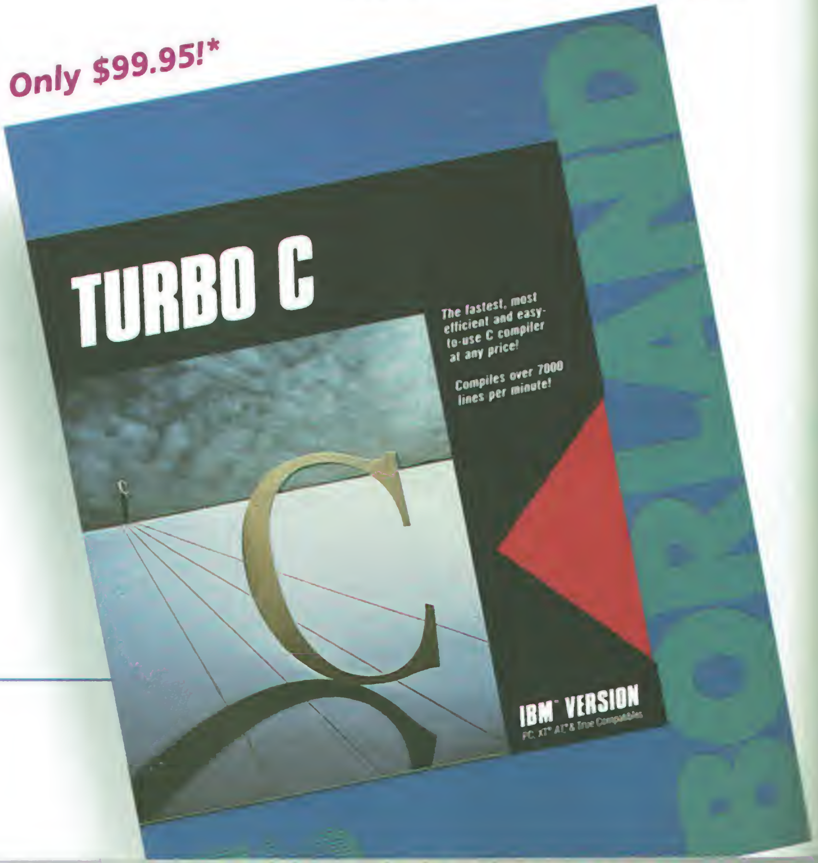
BI-1104

Sieve benchmark (25 iterations)

	Turbo C	Microsoft® C	Lattice C
Compile time	**3.89**	16.37	13.90
Compile and link time	**9.94**	29.06	27.79
Execution time	**5.77**	9.51	13.79
Object code size	**274**	297	301
Price	**$99.95**	$450.00	$500.00

Benchmark run on a 6 Mhz IBM AT using Turbo C version 1.0 and the Turbo Linker version 1.0; Microsoft C version 4.0 and the MS overlay linker version 3.51; Lattice C version 3.1 and the MS object linker version 3.05.

Only $99.95!*


TURBO C
The fastest, most efficient and easy-to-use C compiler at any price!
Compiles over 7000 lines per minute!
C
IBM™ VERSION
PC, XT® AT® & True Compatibles
BORLAND

BORLAND
INTERNATIONAL

Vive la différence

4585 SCOTTS VALLEY DRIVE
SCOTTS VALLEY, CA 95066
(408) 438-8400 TELEX: 172373

TC15

For the dealer nearest you or to order by phone call

(800)255-8008

in CA (800) 742-1133 in Canada (800) 237-1136